UNIVERSIDADE DO MINDELO

DEPARTAMENTO DE CIÊNCIAS ECONÓMICAS E EMPRESARIAIS

CURSO DE LICENCIATURA EM INFORMÁTICA DE GESTÃO

RELATÓRIO DE PROJETO DE LICENCIATURA

ANO LECTIVO 2013/2014 – 4º ANO

Autor: Jorge Michael Gomes Rocha, N.º 2179

Mindelo, 2014

SISTEMA DE GESTÃO DE FILAS DE ESPERA

Relatório apresentado à Universidade do Mindelo como parte integrante do Projeto final do curso de Licenciatura em Informática de Gestão.

Mindelo, 2014

O Orientador,

Eng.º João Dias

DEDICATÓRIA

Dedico este trabalho à toda a minha família, em especial à minha mãe batalhadora que sempre se esforçou incansavelmente para que eu chegasse até aqui e meus padrinhos que me deram todo o apoio no percurso académico.

AGRADECIMENTOS

Agradeço primeiramente a Deus, pela vida, saúde e por me ter dado forças para nunca desistir.

À minha querida avó, Teodora Rocha, que me deu educação e sabedoria durante uma boa parte da minha vida; foi tudo isso que me serviu de base para ser quem sou hoje e onde quer que esteja, sei que está orgulhosa por me ver chegar até aqui.

Aos meus padrinhos Manuel Assunção e Maria Gomes, pelos anos de dedicação, hospitalidade, concelhos, cuidado, amor, carinho e incentivo, porque sem eles não teriam sido possíveis, todos esses anos de estudo.

À minha querida mãe Antónia Gomes, por ter-se dedicado à mim, me encorajando sempre - apesar das dificuldades, ela nunca desistiu - e à toda a minha família, que de uma forma ou de outra, contribuíram direta ou indiretamente para que eu alcançasse mais essa etapa da minha vida.

Ao meu orientador Doutor João Dias, pelo incentivo, apoio, paciência, dedicação e orientação, fundamental para a realização deste trabalho.

Aos meus colegas de curso, pela amizade, convivência e apoio nos momentos necessários, em especial ao meu colega Ivanir Gomes pelos estudos, pelas trocas de ideias fundamentais e pelo seu apoio na transformação das minhas ideias em códigos.

À toda a equipa do Delgraça Mercados, pelo apoio e incentivo, aos meus colegas de trabalho, pelas trocas concedidas; um obrigado especial à Matilde do Rosário pelo apoio na revisão ortográfica.

Por último, a todos os meus amigos, professores e entidades que de alguma maneira contribuíram para que eu conseguisse realizar este trabalho.

A todos, muito obrigado!

RESUMO

Vivemos num mundo competitivo em que o padrão de serviço exigido pelos clientes é cada vez mais elevado. Nos sectores públicos ou privado, os utentes esperam um atendimento acolhedor, rápido e eficiente. Mas isso muitas vezes não acontece pelo facto dos sistemas atuais possuírem apenas vários serviços com senhas diferentes e um *display* onde aparece a senha chamada. Alguns fornecem dados estatísticos e um *display* com vídeos interativos, mas mesmo assim ainda não são capazes de fazer uma gestão eficiente.

Neste trabalho é apresentada uma solução que permite uma melhor gestão da fila e diminuir o tempo de espera através de alertas por meio de SMS, permitindo assim que o cliente não tenha que esperar na fila.

Para realização deste projeto seguiu-se a metodologia de engenharia de *software*.

Para o desenvolvimento do projeto, foi utlizada a linguagem Java na implementação do dispensador, chamador e *display*, e a gestão Web de *back office* foi desenvolvido em PHP e HTML. A base de dados foi desenvolvida em *MySQL*, o sistema foi modelado em UML, e para gestão de conteúdos foi utilizado o *Framework Yii*.

O sistema será capaz de gerir eficientemente uma fila através da qualidade de serviço, disponibilização de conteúdos multimédia, informações úteis, dados estatísticos, serviço de alertas por SMS, atendimento prioritário, distribuição equitativa de trabalho e análise do desempenho de cada funcionário, permitindo uma satisfação total do cliente.

PALAVRAS-CHAVE: Gestão de filas, *Java*, *MySql*, *Yii Framework*, atendimento rápido, alerta SMS, satisfação.

ABSTRACT

We live in a competitive world where the standard of service required by customers is increasingly high. In sectors private or public, the users expect a friendly service, fast and efficient. But this often does not happen because of current systems only having multiple services with different passwords , and a display appears where the password call. Some provide statistical data, and a display with interactive video, but still are unable to make efficient management.

In this work a solution to better management of the queue is displayed, reduce waiting time through alerts via SMS, thus allowing the customer does not have to wait in line.

For realization of this project was followed the software engineering methodology.

To develop the project, was used the Java language implementation of the dispenser, and caller display, and back office management Web was developed in PHP and HTML. The database was developed in MySQL, the system was modeled in UML and Content Management Framework was used Yii.

The system will be able to efficiently manage a queue through quality service, providing multimedia content, useful information, statistics; alerts service SMS, priority seating, fair distribution of work, performance review of each employee, allowing complete satisfaction the client.

KEYWORDS: Queue management, Java, MySQL, Yii Framework, quick service, SMS alert, satisfaction.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

SMS – *Short Message Servisse* (Serviço de mensagens curtas)

BCA – Banco Comercial do Atlântico

INPS – Instituto Nacional de Previdência Social

UML – *Unified Modeling Language* (Linguagem de modelagem unificada)

CPU – *Central Process Unit* (unidade central de processamento)

LED – *Light Emiting Diode* (Díodo emissor de luz)

LCD – *Liquid Cristal Display* (Display de cristal líquido)

FIFO – *First In First Out* (primeiro a entrar, primeiro a sair)

LIFO – *Last In, First Out* (último a entrar, primeiro a sair)

FCFS – *First Come, First Served* (primeiro a chegar, primeiro atendido)

LCFS – *Last Come, First Served* (ultimo a chegar, primeiro atendido)

WWW – *Worl Wide Web* (rede de alcance mundial)

TCP/IP – *Transfer Control Protocol/Internet protocol* (protocolo de controlo de transmissão/ protocolo de internet)

HTTP – *HiperText Transfer Protocol* (protocolo de transferência de hipertexto)

FTP - *File Transfer Protocol* (protocolo de transferência de arquivos)

URL – *Uniform Resource Location* (localizador-padrão de recursos)

PHP – *Personal Home Page* (página web pessoal)

SGBD – Sistema de Gestão de Base de Dados

PERL - *Practical Extraction And Report Language* (Linguagem de relatórios e extração prática)

SQL - *Structured Query Language* (Linguagem de consulta estruturada)

DFD – Diagrama de Fluxo de Dados

DB – *Database* (base de dados)

TMA – Tempo Médio de Atendimento

TME – Tempo Médio de Espera

1. INTRODUÇÃO

A crescente oferta na prestação de serviços a partir da década de 50 tem revelado a importância deste sector na economia mundial. Nesse contexto, percebem-se mudanças de paradigmas principalmente pelo aumento gradual da concorrência, mas também devido a fatores tecnológicos e sociais. O primeiro, devido à evolução tecnológica permitiu novas abordagens como forma de auxílio e gerenciamento da produção e o segundo, pela transformação da conceção de valores pelos clientes, onde estes tornaram-se mais exigentes e informados.

Na medida em que a perceção positiva dos clientes está condicionada a qualidade do atendimento durante a prestação de serviços, alguns fatores durante a prestação de serviços ganham destaque, tais como a cordialidade, acessibilidade, competência e tempo de atendimento.

Sendo assim, este projeto tem como foco principal o desenvolvimento e implementação de um sistema de gestão de filas de espera que não só faça uma gestão eficiente de filas, como também permitia acabar com o exaustivo tempo de espera enfrentado nas empresas, principalmente nas de função pública. Isso será feito através do envio de um alerta por SMS ao cliente avisando-o da aproximação do seu atendimento; assim sendo o cliente poderá aguardar pelo seu atendimento fora da fila, dando-lhe maior comodidade.

Tem-se a certeza que de um serviço como este será aceite por qualquer sector, público ou privado, pois é mais completo do que os sistemas atuais utilizados no país, visto que permite não só gerir a fila em si, mas também gerir os próprios funcionários, a qualidade de serviço, fornecer dados estatísticos importantíssimos para tomada de decisão por parte dos gestores, para além da disponibilização de conteúdos multimédia, com vídeos interessantes, informações úteis, como, estado de tempo, farmácias de serviço, números úteis, etc.

1.1. Motivação

A escolha do tema surgiu de uma conversa com um colega de curso; foi onde teve-se a ideia e este colega acabou por enriquecê-la; ela foi aprovada pelo meu coordenador, que por sua vez prestou uma melhor formulação ao projeto.

Deve-se também ao facto de, como qualquer outro, o autor deste projeto ser um cliente que enfrenta filas diariamente nas empresas; já chegou a passar horas para ser atendido. Por isso há aqui uma grande motivação em ajudar os clientes, na expectativa que o sistema possa diminuir muito o tempo de atendimento ao cliente e que possa ajudar muito as empresas a gerir melhor o seu atendimento.

Às vezes é um teste à nossa paciência, esperar horas e horas, e ver situações que acabam por piorar ainda mais a fila, como por exemplo, quando os funcionários encerram o serviço justamente na hora em que muitas pessoas estão na fila, quando os mesmos funcionários saem para almoçar, ou quando vemos pessoas a "furar" a fila – algo que acontece constantemente nas empresas – quando chega uma pessoa amiga do funcionário, ele sempre dá um jeito de atendê-lo mais depressa.

Após fazer uma análise destas situações, fez-se uma abordagem sobre todas as filas de espera em São Vicente e chegou-se à conclusão que todas elas precisam ser modernizadas com alternativas que possam abrandar o exaustivo tempo de espera nas mesmas, como por exemplo, uma maior atratividade nos *displays*, apresentando informações mais úteis para quem está em espera. É algo que se pensa ser muito inovador: a utilização do SMS para alertar os clientes que se ausentarem da fila, de forma que possam tirar a sua senha e realizar outras tarefas.

Após fazer uma pesquisa, viu-se que este sistema já é uma realidade em algumas empresas no exterior e isso fez surgir no autor uma maior motivação porque, ainda de início ele teve alguma preocupação quanto ao ser possível ou não ter um sistema desse nível; hoje ele está aqui para provar que é mesmo possível.

1.2. Objetivos Do Trabalho

O presente trabalho tem **como objetivo geral:**

Criar um sistema que permita uma melhor gestão no atendimento, através do envio de alertas via SMS.

Objetivos específicos;

Partindo-se de um sistema de filas, explorar e implementar, criar melhorias, adicionar funcionalidades que permitam melhorar o desempenho das referidas filas e fazer com que os seus sistemas de espera passem a ter uma nova visão na ótica dos clientes.

Pretende-se criar um sistema que possa ser utilizado em qualquer empresa, ou seja que permita ser adaptado aos serviços utilizados por cada empresa; sendo assim, facilitará a vida dos clientes que frequentam várias empresas no dia-a-dia.

O sistema deverá permitir:

- Obter em tempo real todos os dados que permitem o controlo e gestão dos funcionários, para a tomada de decisões;
- Envio de mensagens SMS através dum provedor, para alertar os clientes aquando do seu atendimento;
- Melhor gestão para atendimentos prioritários;
- Disponibilização de informações úteis apresentadas no *display*, tais como notícias de atualidade local, estado do tempo, farmácias de serviço, etc.
- Melhoria na qualidade do serviço prestado, melhor aproveitamento dos recursos humanos e aumento de produtividade.

A forma proposta para alcançar estes objetivos é realizar a integração de tecnologias para implementar uma solução que possa atender às necessidades e expectativas dos clientes.

1.3. Metodologia

A metodologia utilizada para realização do projeto foi a engenharia de *software*.

Para realização deste trabalho iniciou-se um conjunto de pesquisas sobre as filas de espera utilizadas nas diversas empresas em Cabo verde, nomeadamente em São vicente, onde visitei as lojas da CVTELECOM, da ELECTRA, do BCA, CAIXA ECONÓMICA, CÂMARA MUNICIPAL, CASA DO CIDADÃO e do INPS.

Durante essas visitas, analisou-se o funcionamento da fila e fez-se o levantamento de algumas informações que tiveram uma grande utilidade neste trabalho.

Foram feitas pesquisas sobre teorias de filas de espera e estudos sobre medidas de desempenho e modelos de filas de espera.

Seguidamente foi feita a análise dos requisitos de funcionante do sistema.

Fez-se a modelagem com diagrama de fluxo de dados, diagrama de casos de uso e diagrama de sequências.

Com o desenho do sistema ficou mais fácil transformar as minhas ideias em código. O primeiro a ser trabalhado foi o dispensador, que é onde se desencadeia todo o processo que inicia com a requisição de um serviço e o levantamento da respetiva senha.

1.4. Dificuldades

O estudo de sistemas de fila de espera não é um tema muito comum do ponto de vista académico, o que torna um pouco difícil encontrar material relacionado nem bibliografia adequada; sendo assim só foi possível encontrar estas informações na internet, onde se exige uma melhor análise, pois estas informações precisaram ser trabalhadas com maior cuidado.

Em termos de programação as filas são estudadas em Algoritmos e Estrutura de Dados, e possuem a mesma lógica de um sistema de filas de espera mas não é demasiadamente aprofundado, o que exigiu um maior estudo em relação às mesmas. A maior dificuldade no trabalho foi a construção de alguns métodos, pois durante o curso não foi possível estudar todos os conteúdos da linguagem java. Estas dificuldades foram superadas graças às vídeo-aulas assistidas durante a fase de programação – o que considero uma importante ferramenta para aprender a programação – e também através das trocas de ideias com o colega Ivanir, em que sempre se chegou a uma solução.

Na implementação do dispensador surgiram várias dificuldades, o que consumiu muito tempo, pois não se poderia avançar para as demais funcionalidades sem que o dispensador estivesse funcionando na perfeição. Uma dessas dificuldades foi na introdução do serviço prioritário, onde surgiu a dúvida, de como seria gerido pelo sistema. A solução encontrada foi separar o serviço normal, de serviço prioritário, sendo assim, há uma senha própria para os serviços prioritários, e influenciam nos dados estatísticos mas de forma separada.

Foi bastante difícil decidir, como seria feita a validação de número de telemóvel de clientes, de modo a evitar que alguém possa usar número alheio de forma abusiva. A melhor forma de fazer isso seria, a autenticação através do telemóvel do cliente com um SMS *request*. Infelizmente não foi possível fazer isso de momento, porque isso exigia um conhecimento em programação para modem, capaz de receber e reenviar um SMS. Mas de qualquer forma foi utilizado a autenticação através de um código gerado pelo sistema no momento em que o cliente faz a sua primeira utilização do serviço SMS.

1.5. Estrutura Do Trabalho

Este documento é composto por cinco capítulos; o primeiro compõe a parte introdutória e metodológica, bem assim a descrição dos objetivos do trabalho a desenvolver.

O segundo capítulo consiste na fundamentação teórica, onde é feita uma abordagem sobre as filas de espera e faz-se entender o funcionamento básico de um sistema de filas, as suas teorias, algoritmos e modelos de filas de espera. Neste mesmo capítulo dá-se uma noção sobre a engenharia de *software* e o respetivo modelo que foi utilizado no projeto; também se faz uma breve abordagem sobre as linguagens de programação utilizadas no projeto.

O terceiro capítulo corresponde à análise de sistemas onde primeiramente é feito o levantamento dos requisitos, seguido da elaboração de diagrama de fluxo de dados e modelação em UML, utilizando diagramas de caso de usos e de sequência; termina-se com o modelo de dados.

O quarto capítulo corresponde à explicação do protótipo desenvolvido, onde é ilustrada a sua implementação e o *layout* das janelas.

O último capítulo corresponde às conclusões do projeto realizado, bem como o trabalho futuro a ser realizado.

2. ENQUADRAMENTO TEÓRICO

Para uma melhor compreensão do tema foi feita uma abordagem sobre o funcionamento de filas, de um modo geral, para ajudar a compreender melhor o que é uma fila de espera em si.

2.1. Noções Básicas Sobre Filas De Espera

Uma fila ocorre sempre que a procura por um determinado serviço é maior que a capacidade do sistema de prover este serviço.

Um sistema de filas pode ser definido como clientes chegando, esperando pelo serviço (se não forem atendidos imediatamente) e saindo do sistema após terem sido atendidos. "Cliente", em teoria das filas, é um termo genérico, aplicando-se não somente a seres humanos. O conceito pode abranger, por exemplo, processos esperando para receber a CPU; pacotes que chegam a um *router* para serem encaminhados; pessoas esperando no caixa do supermercado, etc.

2.1.1. Utilidades de um sistema de filas de espera

O sistema tem uma grande utilidade não só para os clientes mas também para as empresas que adotarem ao sistema.

Para a empresa, espera-se um aumento nos seus negócios, atenderá um maior número de pessoas por dia, haverá uma diminuição nos seus gastos, pois passará a utilizar um sistema local, sem ter que comprar no exterior, onde não haverá gastos de manutenção, entre outros.

Com a utilização do sistema de SMS, a empresa dá possibilidade de cadastrar clientes, isto é aqueles que utilizarem o sistema terão os seus números guardados na base de dados

para futuras utilizações em termos de estatísticas. Os números poderão também ser utilizados pelas empresas para enviar SMS aos clientes em épocas especiais, como no Natal, por exemplo.

Uma melhor disposição dos funcionários para trabalhar, pois quanto menor o número de clientes na fila, menor é a possibilidade de irritação ou estresse por parte dos funcionários.

Os clientes serão os mais beneficiados com um sistema inovador, que permite uma menor espera nas filas, e mesmo que esteja na fila possa desfrutar dum ambiente atrativo, onde o cliente terá acesso à informações importantes tais como farmácias de serviço, estado do tempo na cidade, dicas importantes, e informações sobre a empresa prestadora do serviço.

Caso o cliente tenha a necessidade de se ausentar da fila pode optar por receber um alerta via SMS, avisando sobre a aproximação da sua vez no atendimento. Sendo assim não haverá a necessidade de esperar na fila, pois poderá realizar várias tarefas, até mesmo dirigir-se para outras empresas entrar noutras filas, melhor ainda se for numa empresa que utiliza o mesmo serviço.

2.1.2. Fila sob óptica do cliente

Do ponto de vista do cliente, a sensação de espera é mais importante em sua percepção sobre o serviço que o tempo real que gasta esperando. Essa é uma constatação com implicações gerenciais importantes, pois faz com que o gerente, forçado por restrições de recursos a conviver com filas em seu sistema de serviços pense em meios de atenuar a sensação de espera do cliente. Como o tempo ocioso parece mais longo do que aquele em que estamos ocupados pode-se tentar ocupar o cliente de alguma forma. A música e as revistas em consultórios médicos e dentários são uma tentativa de atenuar a sensação de espera dos clientes. Os bares nos salões de espera de restaurantes, idem.

Outra forma de atenuar a sensação de espera do cliente em fila é dar-lhe a entender que o seu atendimento já se iniciou. Isso se baseia na constatação de que o tempo de espera pelo serviço, da óptica do cliente, divide-se em dois – antes e depois de começar o atendimento e, também, de que a sensação de espera é menos dolorosa na segunda parte. Esse efeito ocorre em restaurantes fast food, que, em horários de pico, escalam funcionários para adiantar a tomada dos pedidos dos clientes ainda na fila, antes de chegarem ao caixa. Embora o objetivo dessa operação seja prioritariamente evitar que o cliente espere chegar ao caixa (ocupando esse recurso) para então decidir sobre o que vai comer, o efeito de atenuar a sensação de espera também está presente.

2.1.3. Aplicações dos sistemas de filas de espera

Existem diversas aplicações da teoria das filas, que podem ser encontradas na literatura de probabilidade, pesquisa operacional e engenharia industrial. Entre elas destacam-se:

- **Fluxo de tráfego** (aviões, carros, pessoas, comunicações)
- **Escalonamento** (pacientes em hospitais, programas em computadores)
- **Prestação de serviços** (bancos, correios, lanchonetes)

2.1.4. Componentes de um sistema de fila de espera

Um sistema de filas consiste no processo de chegada, da distribuição do tempo de serviço, do número de servidores, da capacidade do sistema, da população de usuários e da disciplina de atendimento.

Processo de chegada

O processo de chegada indica qual o padrão de chegada dos clientes no sistema. Apresenta comportamento estocástico, ou seja, as chegadas ocorrem no tempo e no espaço de acordo com as leis da probabilidade; assim, é preciso conhecer qual a distribuição de probabilidade que descreve os tempos entre as chegadas dos clientes.

A distribuição mais comum é a de *Poisson*, ou seja, os tempos entre as chegadas são exponencialmente distribuídos. Entre outras distribuições, estão a de *Erlang*, híper exponencial e arbitrária.

Clientes podem chegar simultaneamente (chegada em *batch*). Se for possível, é necessário também saber a distribuição de probabilidade do tamanho do *batch*. A reação do cliente na fila pode variar. Ele pode esperar independentemente do tamanho da fila, também pode decidir não entrar no sistema caso a fila esteja muito grande (cliente dececionado), ele pode esperar na fila mas depois de um tempo desistir e sair do sistema, e também pode mudar de uma fila para outra em sistemas com servidores paralelos.

O padrão de chegada de clientes em função do tempo pode ser permanente; nesse caso o padrão não muda no tempo, ou seja, a distribuição de probabilidade que descreve as chegadas é independente do tempo. Também pode ser não-permanente, isto é, o padrão de chegada muda com o tempo. Por exemplo, a chegada de clientes diminui no horário de almoço.

Distribuição do tempo de serviço

Assim como no processo de chegada, também é necessário conhecer a distribuição de probabilidade do tempo de serviço, sendo válidas as mesmas distribuições apresentadas.

Os serviços podem também ser simples ou *batch*.

O estado pode ser independente: o processo de atendimento não depende do número de clientes esperando pelo serviço. Em contrapartida, em um estado dependente, o processo de atendimento muda de acordo com o número de clientes na fila. Por exemplo, um servidor pode trabalhar mais rápido quando a fila aumenta ou, ao contrário, ficar confuso e então mais lento.

Da mesma forma que no processo de chegada, o padrão de serviço pode variar de acordo com o tempo. Por exemplo, a experiência adquirida com o serviço pode aumentar a produtividade; o cansaço, por outro lado, pode diminuí-la. Caso não haja variação o padrão é estacionário.

Número de servidores

Esse componente é também conhecido como número de canais de serviço. Indica a quantidade de "pontos de atendimento" do sistema, de forma a servir aos clientes paralelamente. Quando um sistema possui mais de um servidor (multi-servidor ou multicanal), ele pode apresentar duas variações. Em um sistema de fila única, existe uma única fila para todos os servidores, como em um caixa de banco. Em um sistema de múltiplas filas, existe uma fila para cada servidor, como num caixa de supermercado.

Capacidade do sistema

Representa o número máximo de clientes que o sistema suporta, incluindo os que estão em espera e os que estão sendo atendidos. A capacidade pode ser infinita (mais fácil de analisar) ou finita (por exemplo, número limitado de *buffers* em um roteador). Se a capacidade for finita, quando o sistema estiver lotado nenhum cliente pode entrar até que um cliente saia do sistema, liberando espaço.

População de usuários

Este componente indica o número potencial de clientes que podem chegar a um sistema. Pode ser finita ou infinita.

2.1.5. Elementos constituintes de um sistema de filas de espera

Um sistema de gestão de filas é composto por vários componentes de *software* e *hardware*.

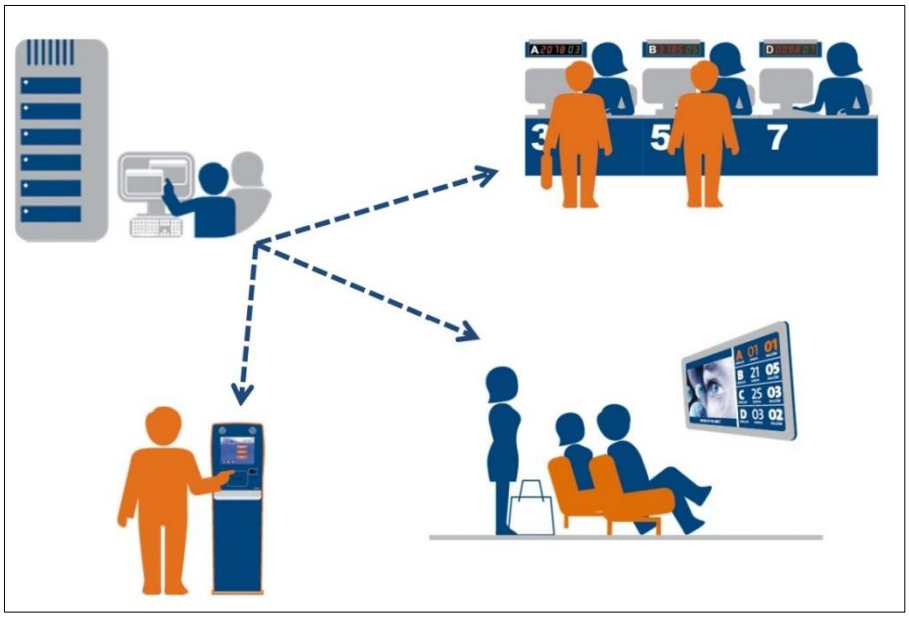


Figura 1 - Componentes de um sistema de Filas de Espera

Dispensador – responsável por disponibilizar as senhas que os utentes solicitam. Pode ser um rolo de senhas pré-impressos ou uma impressora de senhas.

Chamador – componente com a principal responsabilidade de chamar as senhas. É um dispositivo físico (chamador físico) ou uma aplicação utilizada pelos funcionários da organização (chamador virtual).

Display – dispositivo que indica o estado das filas de espera. Pode-se consistir em ecrãs de matrizes de LED ou ecrãs de utilização genérica (LCD, plasmas, etc.). Em alguns casos disponibilizam também informações e/ou conteúdo de entretenimento aos utentes.

Servidor – é responsável pelo controlo geral do sistema, contendo a parte de gestão de *back-office*, a base de dados e o gestor de conteúdos multimédia, e onde se gera valores estatísticos indispensáveis à tomada de decisões por parte dos gestores.

2.1.6. Teoria das filas

A teoria das filas é um método analítico que aborda o assunto por meio de fórmulas matemáticas. Já a simulação é uma técnica que, usando o computador digital, procura montar um modelo que melhor represente o sistema em estudo.

A abordagem matemática de filas se iniciou no princípio do século XX (1908) em Copenhaga, Dinamarca, com A. K. Erlang, considerado o pai da Teoria das Filas, quando trabalhava numa companhia telefónica. Foi somente a partir da segunda guerra mundial que a teoria foi aplicada a outros problemas de filas. Apesar do enorme progresso alcançado pela teoria, inúmeros problemas não são adequadamente resolvidos por causa de complexidades matemáticas (PRADO, 2004).

A teoria das filas envolve o estudo matemático das filas ou linhas de espera. A formação de filas excede a capacidade de fornecer aquele serviço. Os modelos matemáticos se tornam complexos porque normalmente utilizam ferramentas que envolvem um tratamento estatístico ou estocástico. Fornecer uma capacidade excessiva de atendimento gera ociosidade, fornecer um atendimento deficitário gera insatisfação, perda de clientes, perda de produção; tudo isto leva a uma relação muito forte entre as condições de um sistema de filas e a minimização dos custos no atendimento do mesmo.

O estudo de sistemas de filas tem larga utilidade:

- No planeamento e controlo da produção;
- No dimensionamento de sistemas de armazenamento;
- Nos sistemas de transportes;
- Nos sistemas de tráfego (rodo-porto-aéreo-ferroviário);
- Na manutenção de máquinas;
- Em qualquer sistema em que seja provável a formação de filas para determinado atendimento;
- Nos sistemas de saúde;
- Sistemas comerciais.

A figura a seguir ilustra os elementos que compõem uma fila. Nela pode-se observar que numa certa população, surgem clientes que formam fila e que aguardam por um certo tipo de serviço.

Sistema de Filas

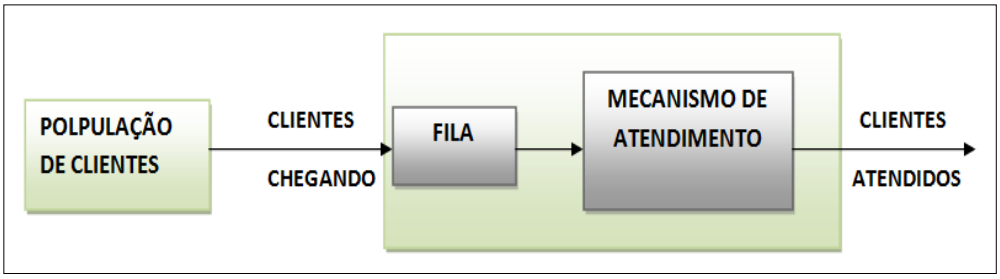


Figura 2 - Sistema de Filas

- A população de clientes pode ser finita ou infinita;
- Os clientes podem chegar um de cada vez ou em blocos;
- A fila pode ter capacidade finita ou infinita;
- O mecanismo de atendimento pode ter um posto ou vários, paralelos;
- O sistema engloba os clientes da fila e os clientes em atendimento.

As figuras a seguir ilustram os tipos de atendimento em filas.

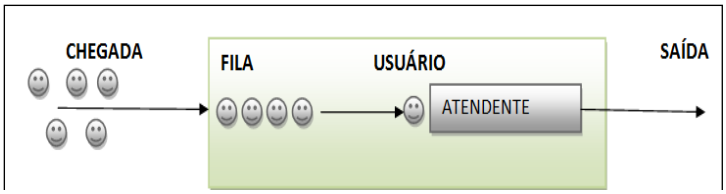


Figura 3 - Fila única, atendedor único

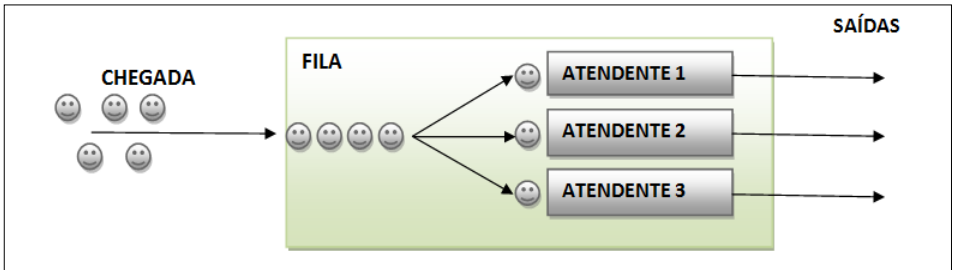


Figura 4 - Fila única, atendedor múltiplos em paralelo

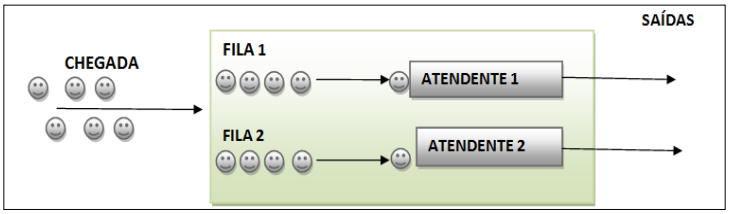


Figura 5 - Filas múltiplas, atendedor múltiplos em paralelo

Figura 6 - Fila única, atendedor múltiplos em paralelo

2.1.7. Características de um sistema de filas de espera

Clientes e tamanho da população - clientes são provenientes de uma população. Quando uma população é muito grande, a chagada de um novo cliente a uma fila não afeta a taxa de chegada de clientes subsequentes e conclui-se dizendo que as chegadas são independentes. Quando a população é pequena, o efeito existe e pode ser considerável.

- **População infinita** => Chegadas independentes
- **População finita** => Chegadas interdependentes

Processo de chegadas - Representa o ritmo de chegadas para a realização de uma atividade e para quantificar as variáveis randómicas (aleatórias) envolvidas, podemos definir:

- **λ**= Ritmo de chegada (exemplo: 4 clientes por minuto)
- **IC** = Intervalo médio entre chegadas (exemplo: 12 segundos)

Processo de atendimento: Representa a realização da atividade e este processo é quantificado pelas variáveis randómicas (aleatórias) definidas a seguir:

- **μ** = Ritmo de atendimento
- **TA** = Tempo de atendimento

Número de servidores (c): Quantidade de servidores que atendem aos clientes.

Disciplina das filas - Trata-se de uma regra que define qual o próximo a ser atendido e o comum é que o primeiro da fila é atendido ou, de uma maneira mais ampla, "o primeiro em chegar é o primeiro a ser atendido" (em inglês, diz-se FIFO: *First in, first out*). Outras disciplinas podem existir, tais como "último a chegar, primeiro a ser atendido" (em inglês diz-se LIFO: *Last in, first out*), serviço por ordem de prioridade, serviço randómico (atendimento aleatório), etc. Portanto, uma característica do cliente pode definir sua prioridade de atendimento

Tamanho médio e máximo da fila (TF) - representa o número de clientes que esperam para ser atendidos ou a média que permanecem na fila.

Disciplina de atendimento

Descreve a forma como os clientes saem da fila de espera para serem atendidos. Algumas disciplinas são:

- **FCFS** (*First Come, First Served*): Primeiro a Chegar, Primeiro a ser Atendido. Disciplina mais comum, inclusive na vida diária.
- **FIFO** (*First In, First Out*): Primeiro a Entrar, Primeiro a Sair).
- **LCFS** (*Last Come, First Served*): Último a chegar, Primeiro a ser Atendido
- **LIFO** (*Last In, First Out*): Último a Chegar, Primeiro a Sair. Aplicável em sistemas em que o item mais recente é mais fácil de ser recuperado, como por exemplo em sistemas de controlo de *stock*.
- **Fila com prioridade**: a cada cliente é atribuída uma prioridade; clientes com maior prioridade têm preferência no atendimento. Pode ser de dois tipos:
  - **Preemptivo**: o cliente com maior prioridade é atendido imediatamente, interrompendo o atendimento ao cliente com menor prioridade. Ao terminar, o cliente de menor prioridade volta a ser atendido, podendo continuar o processo de onde parou ou então reiniciá-lo.
  - **Não-preemptivo**: o cliente com maior prioridade é colocado no início da fila, recebendo o serviço somente quando o cliente em atendimento sai do sistema, mesmo se este for de prioridade mais baixa.
- ***Round-robin* (algoritmo)**: cada cliente recebe uma fatia de tempo do servidor (*quantum*), dentro da qual é atendido. Após o término do *quantum*, se a atividade não foi completada, o cliente é retirado e outro passa a ser atendido. Posteriormente, o cliente que foi interrompido retorna ao servidor e continua a sua atividade. É muito comum em escalonamento de processos da CPU.

2.1.8. Aspetos importantes quanto à administração de filas de clientes

A existência de filas é mau sinal da ótica do serviço ao cliente e bom sinal da ótica estrita da utilização de recursos. O equilíbrio adequado deve ser buscado:

- Deve-se procurar olhar a fila sob a ótica do consumidor para entender suas expectativas, perceções e angústias;
- A espera normalmente pode ser considerada tolerável pelo cliente em horários de pico, mas não a qualquer momento;
- O tempo de espera deve parecer aceitável e razoável;
- A prioridade no atendimento deve ser percebida como justa;
- Deve haver baixa incerteza por parte do cliente quanto ao tempo que terá de esperar;
- Observar cuidadosamente as condições nas quais o cliente terá de esperar;
- Devemos pensar em meios de distrair o cliente para reduzir sua sensação de espera, como dar ao cliente algo para fazer que lhe dê a sensação de que o atendimento já iniciou, prover locais de espera para alguma atividade útil, como o treinamento do cliente;
- Podemos tentar reduzir a aleatoriedade do processo de chegada de clientes utilizando sistemas como o de reservas;
- Podemos mudar o número de servidores para reduzir o tempo de espera em filas;
- Podemos fornecer pontos de atendimento diferenciados, para tipos específicos de serviço, para maior proporcionalidade entre demanda de serviços e tempos de espera.

2.1.9. Medidas de desempenho

A área do conhecimento chamada “teoria das filas” pode auxiliar, por meio de modelos matemáticos normalmente simples de usar, a tomada de decisões para que se consiga o balanço entre os custos da capacidade produtiva e os custos das filas de espera. Essas

decisões podem ser tomadas com base em diversas medidas de desempenho do sistema de filas, como:

- O tempo médio em que cada cliente permanece na fila (importante para a perceção do cliente quanto à qualidade do atendimento);
- O tamanho médio da fila;
- O tempo médio que cada cliente permanece no sistema (fila somada a atendimento);
- O número médio de clientes no sistema (somando-se as filas de clientes no atendimento);
- A probabilidade de ociosidade nas instalações do sistema (importante para dimensionamento de servidores);
- A utilização média dos recursos (importante para a avaliação de produtividade dos recursos);
- A probabilidade de haver determinado número de clientes no sistema (importante para o dimensionamento de espaço necessário para as filas).

Esses parâmetros podem ser calculados com o auxílio da teoria das filas com base em características do sistema, como as distribuições probabilísticas de chegada dos clientes e seus parâmetros (média e dispersão), a configuração de filas escolhidas e o número de servidores. Dessa forma a tarefa de planejar o sistema físico de filas fica facilitado quando, há ao menos uma Ideia a respeito dos dados de entrada necessária.

2.1.10. Análise de sistema de filas em entidades locais

Foi realizado uma análise em algumas filas, em entidades locais, na ilha de são vicente, destacando às com maiores filas, sendo elas, a Caixa Económica de Cabo Verde no centro da Cidade, BCA agência de Praça Nova, CV Telecom, Electra Norte e o Instituto Nacional de Previdência social (INPS) em que durante algum tempo, observou-se o funcionamento e

andamento das filas. Esta análise foi seguida de um inquérito aos gerentes, onde foram efetuadas questões sobre as funcionalidades que possuíam as filas.

Desta análise, constatou-se que o sistema de gestão de filas mais comum nestas entidades é a *Newvision*, criada pela *Tensator*, uma empresa sedeada no Reino Unido, mas representada por Portugal. Eles possuem um conjunto de equipamentos, tais como dispensadores, displays de chamada, painéis de chamada, terminais de chamada e identificadores de mesa. Apesar de ser um sistema bastante completo, não é totalmente aproveitada as suas funcionalidades. Por exemplo temos o caso do BCA e Caixa Económica, utilizam o mesmo sistema mas, no BCA é mais bem aproveitado.

Enquanto no BCA eles utilizam displays de chamadas, na Caixa utilizam painéis de chamada, e no BCA, o sistema é mais bem controlado no *backoffice*.

Na CVTelecom, o sistema é muito bem utilizado, pois há o aproveitamento da maioria das funcionalidades. A Electra Norte, é uma empresa de grande dimensão, muitas vezes contendo enormes filas, mas não possui um sistema eficiente, capaz de amenizar o tempo de espera dos clientes. Utilizam a *Altronix*, um sistema bastante simples, onde utilizam o display de dois dígitos e dispensadores de senhas que utilizam um rolo de senhas que já vêm configurados bem como um chamador de senhas, manual.

Das pesquisas realizadas, o melhor sistema encontrado, foi no INPS, inaugurada no ano passado, um sistema que permite gerir a fila de forma amigável e confortável para os clientes. O sistema possui um conjunto de serviços possíveis e ainda um serviço prioritário. Possui ainda a solução de *Tv corporate* com uma tela de elevada robustez, dinâmica e personalizada, com informações bastante úteis para os clientes.

O que se pode concluir, é que estes sistemas, são usadas mais para controlar, do que, para gerir as filas, pois o sistema devia ser melhor aproveitado em termos de gestão a nível geral. Poderia haver um melhor aproveitamento dos *displays*, não só para apresentar os conteúdos da fila, mas também para passar informações úteis sobre a entidade e também assuntos que podem interessar os clientes.

2.2. Engenharia De *Software*

O desenvolvimento de um *software* é um processo de aprendizagem social [Baetjer, 1998], um diálogo em que o conhecimento que se deve transformar em *software* é reunido e incorporado no sistema. O processo fornece interação entre utilizadores e engenheiro de sistemas, entre utilizadores e tecnologia (ferramentas evolutivas), e entre engenheiro de sistemas e tecnologias.

Na realidade, desenvolver *software* é um processo de aprendizagem interativo e o resultado é uma corporização de conhecimento recolhido, destilado e organizado à medida que o processo é conduzido.

No processo de desenvolvimento de *software*, é utilizado um modelo onde o objetivo é proporcionar ao projeto uma estrutura que reduza os riscos e incertezas, e de aumentar a governabilidade do processo de desenvolvimento.

Nos pontos seguintes irei descrever o modelo escolhido para o desenvolvimento do meu Projeto.

2.2.1. Modelo em cascata

O **modelo em cascata** é um modelo de desenvolvimento de *software* sequencial no qual o desenvolvimento é visto como um fluir constante para frente (como uma cascata) através das fases de análise de requisitos, projeto, implementação, testes (validação), integração, e manutenção de *software*.

A origem do termo *cascata* é frequentemente citado como sendo um artigo publicado em 1970 por **W. W. Royce**; ironicamente, Royce defendia um abordagem iterativa para o desenvolvimento de *software* e nem mesmo usou o termo *cascata*. Royce originalmente descreve o que é hoje conhecido como o modelo em cascata como um exemplo de um método que ele argumentava *ser um risco e um convite para falhas*.

Este modelo propõe uma abordagem sistemática, linear e sequencial ao desenvolvimento de *software*, onde se iniciam ao nível do sistema de informação e prosseguem com as fases de requisitos de software, desenho do sistema, desenho dos programas, codificação, testes e manutenção.

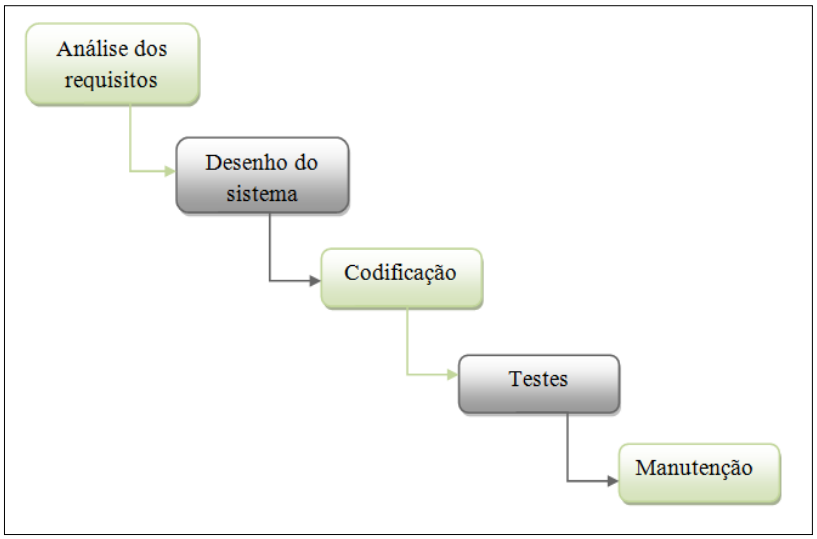


Figura 7 - Modelo em cascata

Analise dos requisitos - Trata-se de uma fase fundamentalmente de engenharia e modelação do sistema de informação. O *software* é sempre o subconjunto de um conjunto (sistema) maior; por isso, o trabalho inicial consiste no estabelecimento das necessidades e requisitos globais de informação da organização e na subsequente atribuição de um subconjunto ao *software*. O processo de determinação dos requisitos é, de seguida, intensificado, concentrando-se especificamente no *software* a construir.

É necessário que os engenheiros de sistemas (analistas) compreendam bem o domínio aplicacional nas suas diversas vertentes – informação, funções, comportamento, desempenho e interfaces.

Desenho do sistema - Esta fase, denominada igualmente de desenho lógico, visa modelar uma solução de um modo independente da tecnologia, isto é, baseada nas funções do sistema a informatizar e não na tecnologia em que o sistema será implementado. O desenho dos programas traduz os requisitos num modelo de *software* cuja qualidade pode ser avaliada antes de se proceder à codificação.

Codificação - A transformação de um projeto para um código deve ser a parte mais evidente do trabalho da engenharia de *software*. O desenho físico tem de ser traduzido para uma linguagem própria da máquina.

Testes - Os testes iniciam-se após o código ter sido escrito. Os testes concentram-se na lógica interna do *software* (programas bem feitos?) e nas funções externas (satisfazem os requisitos do utilizador?).

Operação/Manutenção - Após ser aceite pelo utilizador, o *software* tem de ser instalado no ambiente operacional para que foi construído. No entanto, após a instalação haverá sempre lugar a alterações ao *software*, devido quer a erros que passaram na malha dos testes, quer a alterações ao ambiente de negócio que o software visa servir, quer a alterações no ambiente tecnológico subjacente (hardware, sistema operativo, etc.). Essas alterações são efetuadas no âmbito do suporte e manutenção ao sistema, após a instalação.

2.3. Algoritmo Fila

Em algoritmos e estrutura de dados, uma fila é definida por uma sequência ordenada de elementos, podendo ser adicionados novos elementos numa das extremidades e retiradas na outra extremidade.

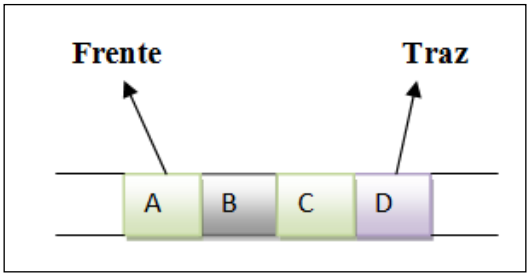


Figura 8 - Inserção dum elemento numa fila

Regras

- Só os elementos que estão à frente é que podem ser retirados
- A fila de espera comporta-se como um FIFO (“First-IN, First-Out”)
- A fila de espera mantém a ordem dos elementos

Operações na fila

- ***createQueue*** **–** cria a fila.
- ***destroyQueue*** **–** coloca a fila inacessível.
- ***emptyQueue*** **–** verifica se a fila está vazia.
- ***fullQueue*** **–** verifica se a fila está cheia.
- ***enQueue*** **–** adiciona um item no fim da fila.
- ***deQueue*** **–** remove e devolve o item no início da fila.

Deslocamento da fila

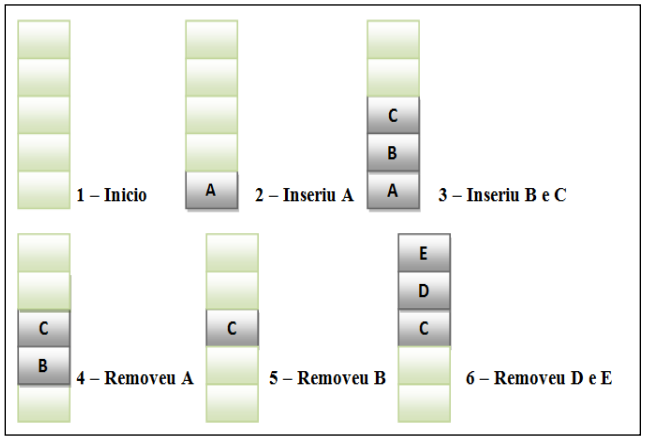


Figura 9 - Deslocamento de fila

1) front = 0, rear =-1

2) front = rear = 0

3) rear = 2, rear = 0

4) rear =2, front = 1

5) front = rear = 2

6) rear = 4, front = 2

Exemplo de Implementação de uma classe fila

```
class Queue {
        private int front, rear;
        private int maxelts;
        private Object [] elts;

        public Queue (int tamanho) {
                elts = new Object[tamanho];
                front = 0;
                rear = 0;
                maxelts = tamanho;
        }

        public boolean empty () {
                return (front == rear ? true : false);
        }
```

```
        public boolean full () {
                if ((rear + 1) % maxelts  == front)
                        return true;
                else return false;
        }

        public void enqueue (Object it) {
                rear = ++rear % maxelts;
                elts [rear] = it;
        }

        public Object dequeue () {
                front = ++front % maxelts;
                return (elts [front]);
        }
}
```

Figura 10 - Exemplo de implementação de uma classe fila

2.4. Ambiente e Desenvolvimento

Neste tópico são enumeradas as principais linguagens de programação utilizadas no sistema bem como as ferramentas que serviram de suporte para o desenvolvimento.

2.4.1. Linguagens de programação

Uma das principais metas das linguagens de programação é permitir que programadores tenham uma maior produtividade, permitindo expressar suas intenções mais facilmente do que quando comparado com a linguagem que um computador entende nativamente (código de máquina). Assim, linguagens de programação são projetadas para adotar uma sintaxe de nível mais alto, que pode ser mais facilmente entendida por programadores humanos. Linguagens de programação são ferramentas importantes para que programadores e engenheiros de *software* possam escrever programas mais organizados e com maior rapidez.

A principal linguagem utilizada no meu projeto foi a linguagem java, que será abordada logo abaixo.

Java

Java é uma linguagem de programação desenvolvida por James *Gosling*, juntamente com outros colaboradores, no início da década de 1990, na empresa *Sun Microsystems*.

A linguagem de programação Java é orientada a objetos (comportamento dos objetos determinados por classes) e compilada em *bytecode* (as instruções são executadas através de uma Máquina Virtual Java - JVM e podem ser processadas em sistemas com suporte a C++). A sintaxe da linguagem Java é similar às linguagens C e C++.

As principais características de Java são as seguintes:

- **Concisa e simples** - Não contém redundâncias e é fácil de entender, implementar e usar. Parecida com C++ para facilitar compreensão por grande parte de programadores. É uma evolução de C++: não suporta aritmética de ponteiros, registros, etc.

- **Orientada a objetos** - Suporta os principais conceitos de orientação a objetos. Favorece extensibilidade e re-usabilidade.

- **Provê acesso a Internet/WWW** - Contém bibliotecas especiais que possibilitam o trabalho com protocolos TCP/IP como HTTP e FTP. Permite acesso às URL.

- **Robusta** - Fortemente tipada. Programas são confiáveis. Reduz imprevistos em tempo de execução: variáveis são automaticamente inicializadas, uso disciplinado de ponteiros, rotinas devem ser chamadas corretamente, etc.

- **Portáveis** - Aplicações funcionam do mesmo jeito em qualquer ambiente. Completamente especificada. Não contém aspetos dependentes da implementação: o tamanho dos tipos é fixo para qualquer implementação, etc.

- **Segura** - Restrições de acesso a arquivos (*applets*), manipulação de ponteiros, etc. Implica que não é útil para desenvolver certas aplicações como `*device drivers'*, etc.

- **Concorrente** - Suporta aplicações concorrentes: *multithreads* e monitores.

- **Independente de plataforma** - Código gerado pelo compilador funciona em qualquer ambiente. Geração de *bytecode* que pode ser interpretado para qualquer arquitetura e sistema operacional tendo o sistema Java. Facilita distribuição de *software*.

- **Interpretada** - Facilita desenvolvimento exploratório. Perde em eficiência.

- **Compilada** - Utilizando compiladores, *bytecodes* podem ser traduzidos em tempo de execução para código de máquina.

No projeto, a linguagem Java foi utilizada na implementação do dispensador, do chamador, e do *display* de apresentação.

PHP

Linguagem desenvolvida por *Rasmus Lerdorf* em 1994; a primeira versão do 2 PHP tinha como função monitorar as pessoas que acessavam o seu *site*.

PHP conhecido como (*Hypertext Preprocessador*), adquirindo maior funcionalidade é representado seriamente na área de desenvolvimento da ***web***, passando assim por melhoramentos indicando assim sua aplicação pré processador de hipertexto.

Atualizado em 1997, o PHP teve melhorias diversas, entre elas inclusão de novas funcionalidades como suporte a várias bases de dados comerciais.

As principais características do PHP são:

- **Manipulação de bases de dados PHP** - Oferece *interfaces* para a maioria dos SGBD comerciais.
- **Manipulação de arquivos** - Onde podem realizar qualquer tipo de operação, como criar, modificar, mover e apagar dados através de comandos do próprio código.
- '**Funções de correio eletrônico** - enviar e receber mensagens através do seu próprio servidor *web*.
- **Sintaxe similar a Linguagem C/C++ e o PERL** - muito mais fácil para criar um ambiente *web*.

A linguagem PHP, realiza várias funções embora tipicamente utilizada em sistemas operacionais como Linux, Free BSD, e também sistema operacional Microsoft Windows.

- **Portabilidade** - tem independência de plataforma.

Yii framework

Yii é um *framework* de alta performance em PHP que utiliza componentes para o desenvolvimento de grandes aplicações *Web*. Permite máxima reutilização de códigos na programação *Web* e pode acelerar significativamente o processo de desenvolvimento. O nome *Yii* (pronunciado i) representa as palavras fácil (*easy*), eficiente (efficient) e extensível (*extensible*).

Eis suas principais características:

- **Rápido**

A arquitetura do *Yii* permite que o mesmo carregue somente o necessário para aplicação no presente momento e, em conjunto com o suporte a cache (APC), consegue um RPS (Requisição por Segundo) bastante interessante.

- **Seguro**

Um *framework* com um excelente desempenho não é nada se a sua segurança é fraca.O *Yii* por padrão faz o tratamento/validações de entrada e saída de dados, oferece suporte a ataques SQL *Injection*, *Cross-site scripting* (XSS), CSRF (*Cross-Site Request Forgery*), *Cookie Attack* e entre outros.

- **Profissional**

Yii segue o padrão MVC na sua estrutura, garantindo a separação entre camadas lógicas e camadas de apresentação do projeto a ser desenvolvido. Ele auxilia o desenvolvedor a ter um código mais limpo e reutilizável (DRY – *Don't repeat yourself*) sem grandes esforços, supre a elaboração de sites simples bem como aplicações extremamente complexas, sendo necessário se preocupar apenas com tarefas específicas do projeto.

2.4.2. Base de dados

Bancos de dados ou bases de dados são coleções organizadas de dados que se relacionam de forma a criar algum sentido(Informação) e dar mais eficiência durante uma pesquisa ou estudo. São de vital importância para empresas, e há duas décadas se tornaram a principal peça dos sistemas de informação.

A base de dados utilizada no projeto foi o MySql.

MySQL

A busca por sistema que sejam mais rápidos, preciso e que automatizem o maior número de funções possíveis, foi que gerou os modernos sistemas de gestão de base de Dados.

O MySQL se destaca primeiro por ser desenvolvido pela colaboração de sua comunidade, através do modelo de Software Livre, mas também pela posição de destaque que ele ocupa em sistemas *web*, sendo a base de dados mais utilizado na *internet*. Mas as funcionalidades do MySQL são ainda muito subutilizadas pelos sistemas *web*, uma vez que os programadores *web* ainda não dominam a linguagem SQL e as ferramentas desta poderosa ferramenta de gestão de Base de Dados.

Apache

O servidor Apache ou Servidor HTTP Apache, é o mais bem sucedido servidor web livre. Os motivos incluem a sua excelente performance, segurança, compatibilidade com diversas plataformas e todos os seus recursos.

O servidor Apache (ou *Apache Server*) surgiu no *National Center of Supercomputing Applications* (NCSA) através do trabalho de *Rob McCool*. Ao sair da NCSA, McCool parou de trabalhar no software (que nessa época recebia justamente a denominação NCSA) e então várias pessoas e grupos passaram a adaptar o servidor Web às suas necessidades. No entanto, foram *Brian Behlendorf* e *Cliff Skolnick* os principais responsáveis pela retomada do projeto, contando logo em seguida com o apoio de *Brandon* Long e Beth Frank. Estes últimos tinham a tarefa de continuar com o desenvolvimento do servidor, mas pela NCSA. Não demorou muito para que eles se juntassem ao Apache *Group*.

Principais características do Apache

- Possui suporte a *scripts cgi* usando linguagens como Perl, PHP, Shell Script, ASP, etc.;
- Suporte a autorização de acesso podendo ser especificadas restrições de acesso;

- Separadamente para cada endereço/arquivo/diretório acessado no servidor;
- Autenticação requerendo um nome de usuário e senha válidos para acesso a alguma;
- Página/subdiretório/arquivo (suportando criptografia via *Crypto* e MD5);
- Negociação de conteúdo, permitindo a exibição da página Web no idioma
- Requisitado pelo Cliente Navegador;
- Suporte a tipos mime;
- Personalização de *logs*;
- Mensagens de erro;
- Suporte a virtual *hosting* (é possível servir 2 ou mais páginas com endereços/ portas diferentes através do mesmo processo ou usar mais de um processo para controlar mais de um endereço);
- Suporte a IP virtual *hosting*;
- Suporte a *name virtual hosting*; Suporte a servidor *Proxy ftp e http*, com limite de acesso, *caching* (todas flexivelmente configuráveis);
- Suporte a *proxy* e redirecionamentos baseados em URL para endereços Internos;
- Suporte a criptografia via SSL, Certificados digitais;
- Módulos DSO (*Dynamic Shared Objects*) permitem adicionar/remover funcionalidades e recursos sem necessidade de recompilação do programa.

2.5. Ferramentas Utilizadas

Antes de começar a falar sobre as ferramentas utlizadas no meu projeto, vou falar um pouco dos *softwares* livres, sendo que todos os softwares que utilizei são ferramentas que estão disponíveis gratuitamente na internet.

Software Livre é uma forma de manifestação de um *software* em que, resumidamente, permite-se adaptações ou modificações no seu código de forma espontânea, ou seja, sem que haja a necessidade de solicitar permissão ao seu proprietário para modificá-lo. Seus objetivos concedem aos usuários a liberdade de controlo na execução e adaptação a sua computação e processamento de dados às suas necessidades

(concessão plena liberdade de controle e independência, através da disponibilidade de código fonte para análise e alterações); bem como permitindo-lhes a liberdade social, para ser capaz de cooperar ativamente com todos os usuários e desenvolvedores de sua escolha.

Os usuários de *software* livre estão livres dessas atividades, porque eles não precisam pedir qualquer permissão, eles não estão restritos nas atividades por meio de licenças proprietárias restritivas (por exemplo, cópia restrita), ou requisitos de ter de concordar com as cláusulas restritivas dos outros (por exemplo, acordos de não divulgação), e eles não estão restritos desde o início (por exemplo, através deliberada a não disponibilidade de código fonte).

MySQL workbench

Pois bem, o MySQL *Workbench* é uma ferramenta com *interface* gráfica que permite modelar base de dados, foi criado pela MySQL AB e é sem dúvida uma ferramenta muito completa e útil para quem trabalha com base de dados.

Principais funcionalidades

- Criação de diagramas EER
- SQL Scripts
- Catálogo da Base de dados
- Visualização total do *Dashboard*
- Informações sobre o objeto selecionado
- Gestão de ligações a servidores MySQL
- *Forward Engineering* e Engenharia Reversa

Netbeans

O mundo da programação de aplicações está em constante mudança. Se há uns anos atrás uma aplicação nativa era o ideal para correr numa máquina, hoje em dia a vertente Web e os dispositivos móveis mudaram por completo todo o paradigma associado ao desenvolvimento de aplicações.

Quando se fala em plataformas para desenvolvimento, o *Netbeans* 8.0 da Oracle é sem dúvida uma das mais robustas, completas e populares pois dá suporte a um conjunto vasto de tecnologias, tais como:

- Java
- Java EE
- JavaScript
- HTML5
- PHP

2.6. Módulo SMS

Uma das funcionalidades mais importantes deste projeto é a utilização de alertas por SMS, que se considera ser uma grande inovação para os sistemas de filas de espera em Cabo Verde. A utilização de alertas por SMS é uma das principais ações que prometem acabar com filas e permitir uma melhor gestão no atendimento.

SMS é a sigla de ***Short Message Service***, que em português significa **Serviço de Mensagens Curtas.**

O SMS é um serviço disponível tanto em telemóvel quanto via internet, podendo ser enviado de qualquer lugar do mundo, independente da operadora do telemóvel, do local onde a pessoa se encontra; é um dos meios de comunicação mais utilizados hoje em dia, além das redes sociais.

2.6.1. Envio de mensagens através de *gateway*

Um *gateway* SMS permite ao computador enviar ou receber SMS e a transmissões é feita a partir de uma rede de telecomunicações.

A conexão com a rede móvel é feita através da aquisição de um número de cartão SIM da operadora de telefonia móvel e instalá-lo no *gateway*. Normalmente, os aparelhos de *gateway* são utilizadas para centenas de milhares de mensagens de texto por mês.

Um *gateway* SMS normalmente fica entre o usuário final que precisa para enviar / receber SMS e SMSC de uma rede móvel. Esses *gateways* fornecem uma escolha de protocolos, incluindo HTTP, SMTP, SMPP e serviços da Web. Os prestadores de serviços de *gateway* SMS incluem agregadores SMS e operadoras de telefonia móvel.

Para testes, foi utilizado o serviço *Twilio Gateway*, disponível gratuitamente na internet em **https://www.twilio.com/**, onde pode-se criar uma conta e testar gratuitamente o envio de SMS durante um período de tempo. O envio de SMS a partir do sistema é feito através do Twilio jar que é adicionado no código Java, e o sistema é configurado de modo que possa utilizar as bibliotecas seguintes:

```
import com.twilio.*;
import com.twilio.sdk.*;
import com.twilio.sdk.resource.factory.*;
import com.twilio.sdk.resource.instance.*;
```

3. ANÁLISE DE SISTEMAS

Neste capítulo faz-se o levantamento dos requisitos funcionais e não funcionais do sistema e mostra-se o funcionamento das principais ações por meio de diagramas de fluxo de dados, *use case* e sequencias, bem como o modelo de dados.

3.1. Levantamento Dos Requisitos

Um requisito funcional define uma função de um *software* ou parte dele. Ele é o conjunto de entradas, seu comportamento e sua saída, ou seja, envolve cálculos, lógicas de trabalho, manipulação e processamento de dados, entre outros. Dentro dos requisitos funcionais também encontram-se a arquitetura do aplicativo, diferentemente da arquitetura técnica, que pertence aos requisitos não funcionais.

Requisitos não funcionais são relacionados ao uso da aplicação em termos de desempenho, usabilidade, confiabilidade, disponibilidade, segurança e tecnologias envolvidas. Muitas vezes, os requisitos não funcionais acabam gerando restrições aos funcionais.

Entender profundamente os tipos de requisitos no momento de definir seu *software* pode decidir o futuro dele. Por outro lado, também é preciso ter em mente que o *software* pode ser beneficiado se houver um pouco de flexibilidade no seu desenvolvimento. Assim, unindo os conceitos, é possível ter um sistema de qualidade sem um alto custo de aquisição.

Adiante segue-se a listagem dos requisitos funcionais e não funcionais do sistema.

3.1.1. Requisitos funcionais do sistema

- **Segurança e Controle de Acessos** – o sistema tem que possuir um controlo de acesso para cada usuário com níveis e privilégios diferentes. Por exemplo, o

funcionário só terá acesso à uma parte das funcionalidades, enquanto o administrador terá acesso a tudo.

- **Cadastro de Usuários, Perfis e Acessos** – o sistema deverá possuir cadastro para usuários, onde terão um nome de utilizador e palavra-chave, podendo estes alterá-lo a qualquer momento.
- **Cadastro de Funcionários** – o sistema deverá possuir um cadastro para os funcionários do sistema e permitir a introdução dos seus dados de identificação.
- **Validação de clientes** – o sistema deverá possuir um método que faça a validação dos clientes, por meio de um código que confirma o seu número de telemóvel;
- **Envio de SMS** – o sistema deverá possuir uma *interface* que permita ao cliente solicitar o envio de alertas por SMS;
- **Gestão de estratégias e de serviços** – deverá permitir a atribuição dos serviços por balcão de acordo com a estratégia desejada. Por exemplo, utilizar os horários para atribuir um serviço principal e um serviço secundário a um determinado operador de balcão;
- ***Reset* no final do dia** - o sistema deverá configurar o sistema para reiniciar automaticamente no final do dia, os contadores e limpar as filas;
- ***Restart* em caso de falha de energia** - Numa falha de energia toda a informação é armazenada no banco de dados e após a inicialização do servidor o sistema será reiniciado de onde parou antes;
- **Definições de Prioridades** – Os serviços podem ter diferentes níveis de prioridades entre eles.
- **Informações estatísticas atualizadas** - múltiplas métricas de análise. Estatísticas de serviço, utilizador, mesa, tempo médio de serviço, o tempo máximo de serviço, etc., relatórios e capacidade de exportar param diferentes formatos;
- **Indicadores em tempo real** - Recolha de dados em tempo real; visualização da evolução da fila, os tempos reais de espera, etc. Também disponível visualizar em gráfico;

3.1.2. Requisitos não funcionais do sistema

- **Implementação** – o sistema deverá ser desenvolvida nas linguagens, Java, PHP e *javaScript*.
- **Bancos de Dados** – os dados deverão ser guardados e geridos numa base de dados previamente construída em MySql.
- **Servidor de Aplicação** - o sistema e a base de dados deverão ser suportados pelo servidor *Apache*.
- **Navegadores** – o sistema web pode ser aberto em qualquer navegador.
- **Portabilidade** – o sistema poderá ser transportado e instalado em outro computador e rodar em qualquer plataforma;
- **Facilidade e confiabilidade** – o sistema deverá ser de fácil acesso para os usuários, principalmente para os clientes e deverá transmitir credibilidade aos mesmos, em relação ao uso de SMS.

3.2. Arquitetura Técnica

A estrutura do meu sistema segue o desenho geral de um sistema de filas de espera comum, que contem o dispensador, computadores com a aplicação chamador, o *display* para apresentação de conteúdos e o servidor com a parte administrativa do sistema.

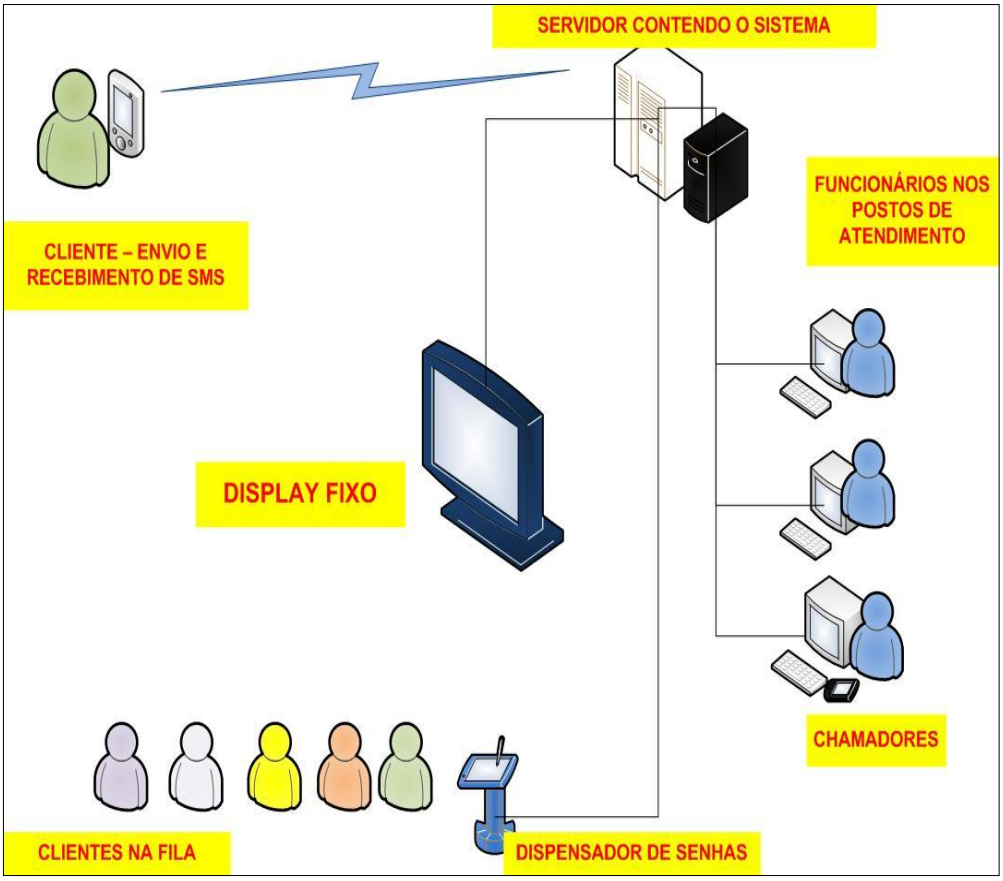


Figura 11 - Arquitetura geral do sistema

3.3. Arquitetura Funcional

A maioria dos problemas encontrados em sistemas orientados a objetos tem sua origem na construção do modelo, no desenho do sistema. Muitas vezes as empresas e profissionais não dão muita ênfase a essa fase do projeto, mas é uma das melhores fases; no meu caso me ajudou muito a arquitetar melhor o sistema por completo.

Comecei por fazer uma análise através do diagrama de fluxo de dados:

3.3.1. Diagrama de fluxo de dados

O Diagrama de Fluxo de Dados (DFD) é uma das principais ferramentas utilizadas no projeto de sistemas de informação. O DFD é um diagrama gráfico baseado apenas em quatro símbolos, que mostra a estrutura do sistema e sua fronteira, ou seja, todas as relações entre os dados, os processos que transformam esses dados e o limite entre o que pertence ao sistema e o que está fora dele.

DFD é uma representação em rede dos processos (funções) do sistema e dos dados que ligam esses processos. Ele mostra o que o sistema faz e não como é feito. É a ferramenta de demonstração central da análise estruturada.

Listagem de eventos

- Cliente consulta o estado de serviço e de senha
- Cliente obtém uma senha
- Cliente pede notificação por SMS
- Cliente retira a sua senha
- Funcionário (atendedor) chama nova senha
- Funcionário (*Admin*) Cria, modifica e elimina balcão
- Funcionário (*Admin*) Cria, modifica e elimina Serviço
- Funcionário (*Admin*) Cadastra novo funcionário
- Funcionário (*Admin*) consulta estado de fila
- Dispensador emite a senha
- Dispensador apresenta informações da fila

Os diagramas seguintes apresentam as partes componentes do um sistema e as *interfaces* entre elas. É um conjunto integrado de procedimentos, e os eventos entre eles. A figura 12 apresenta o DFD nível 1 do sistema.

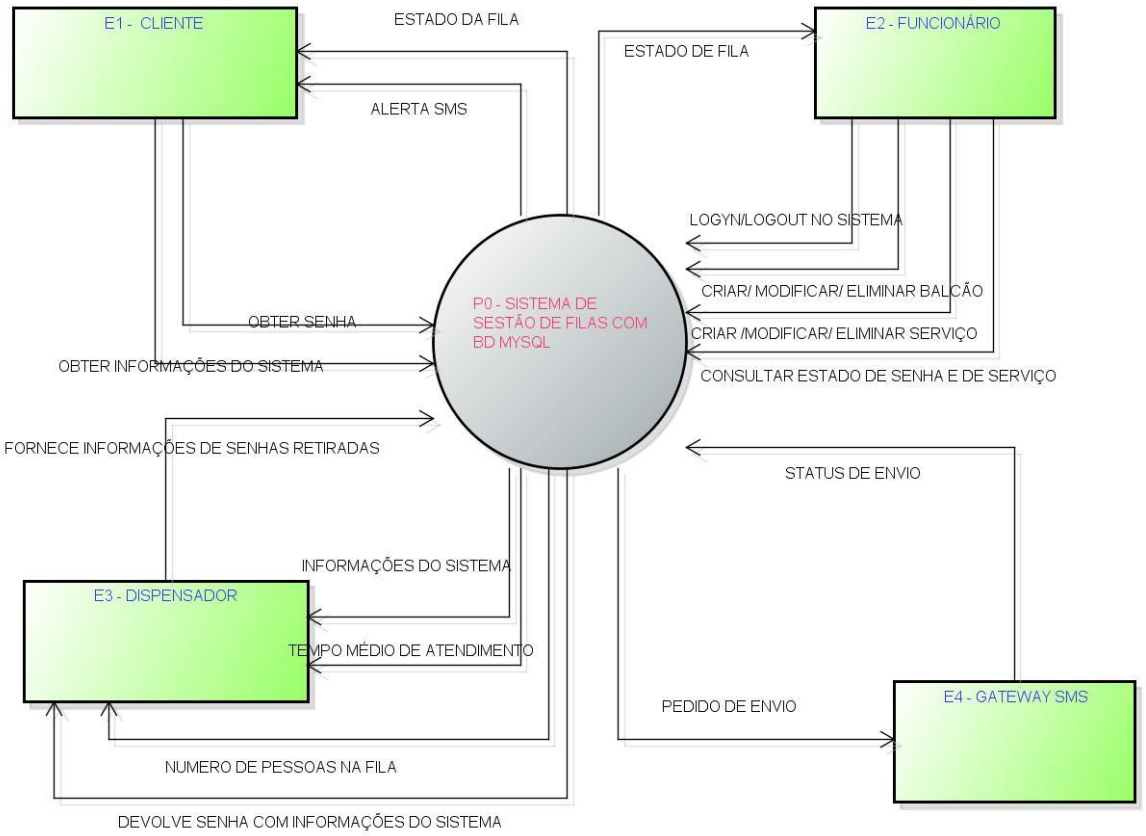


Figura 12 - DFD nível 1-Sistema de Gestão de filas

A figura abaixo por sua vez mostra um DFD mais detalhado (Nível 2), que permite compreender melhor o sistema. Como pode-se observar na figura 13, o funcionário pode cadastrar como administrador, e com essa permissão já pode cadastrar um novo funcionário, criar, modificar e eliminar serviços e balcões e também tem acesso à todas as funcionalidades do sistema.

O cliente, apenas pode pedir senha, pedir alerta por SMS e consultar estado de senha e de serviços no dispensador e visualizações no display.

O dispensador emite a senha e apresenta informações de senha e de serviços, e o Gateway SMS é o responsável por enviar o SMS.

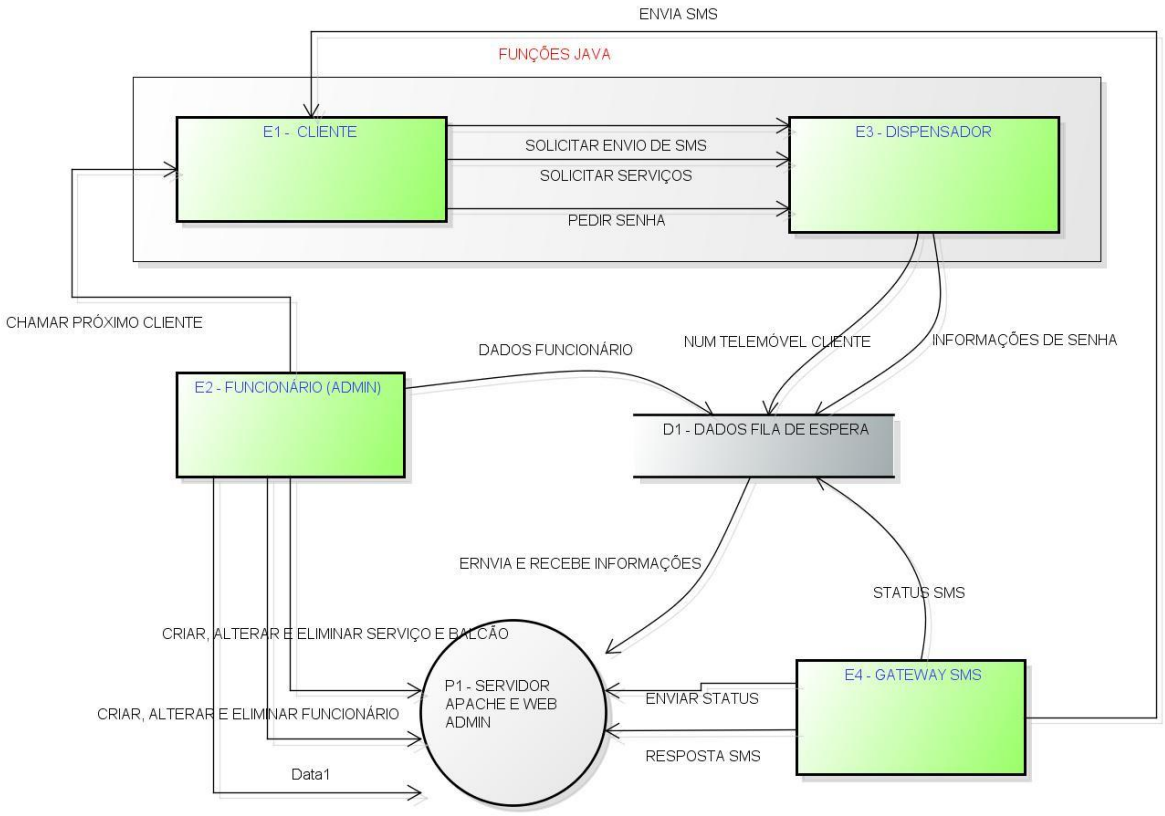


Figura 13 - DFD Nível 2 sistema de filas

3.3.2. Diagramas de casos de uso

O diagrama de caso de uso descreve a funcionalidade proposta para um novo sistema que será projetado e é uma excelente ferramenta para o levantamento dos requisitos funcionais do sistema. Achou-se conveniente utilizá-la neste projeto para mostrar a funcionalidade dos principais eventos do sistema.

Casos de Uso – Gestão de Senhas

O caso de uso Gestão de senha representada na figura 14, consiste num conjunto de operações que serão executadas pelo funcionário de uma determinada empresa, que será feita através dum chamador, e os clientes que fazem a obtenção da senha através dum dispensador.

- Obter senhas – escolher um determinado serviço, selecionar as opções desejadas e esperar pela impressão do *ticket*;
- Chamar nova senha – chamar a próxima senha para um determinado serviço;
- Colocar senha em espera – suspender a senha por um determinado período de tempo, caso seja necessário executar tarefas que exigem algum tempo mas que não seja necessário o pedido de uma nova senha (por exemplo, o preenchimento de um formulário), enquanto isso o funcionário pode chamar uma nova senha;
- Transferir senhas – transferir uma senha para outro serviço;
- Cliente ausente – Indicar que o cliente não está presente no momento da sua chamada à fila.

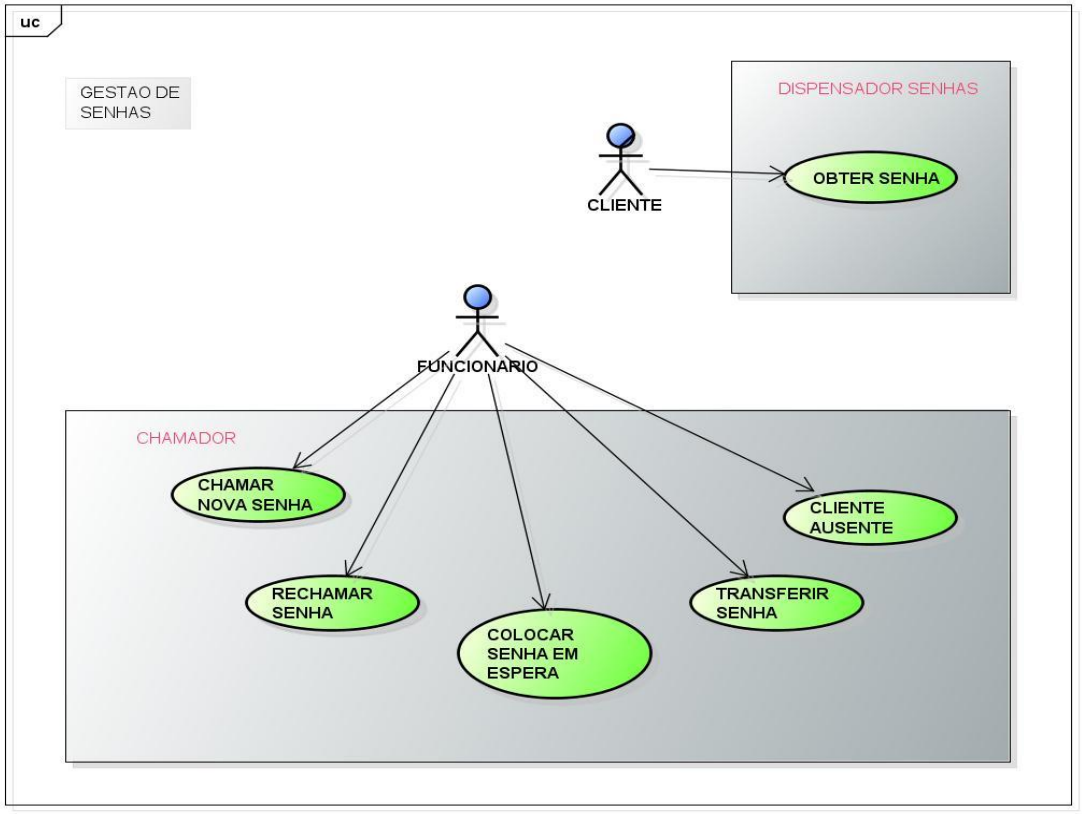


Figura 14 – Caso de Uso gestão de Senhas

Casos de uso – Gestão de serviços

As informações serão executadas por um funcionário do sistema, com permissões de administrador e permite criar, modificar e eliminar serviços do sistema, tal como mostra a figura 15. Um funcionário, ao encerrar um serviço, temporariamente pode transferi-lo para outro funcionário, que continuará o atendimento.

Os serviços são criados pelo administrador, e ele é que decide qual o funcionário a ocupar cada serviço. Além disso um funcionário pode ao mesmo tempo atender em dois ou mais serviços, isso no caso de não haver mais clientes num determinado serviço e passar a chamar naquele que houver fila.

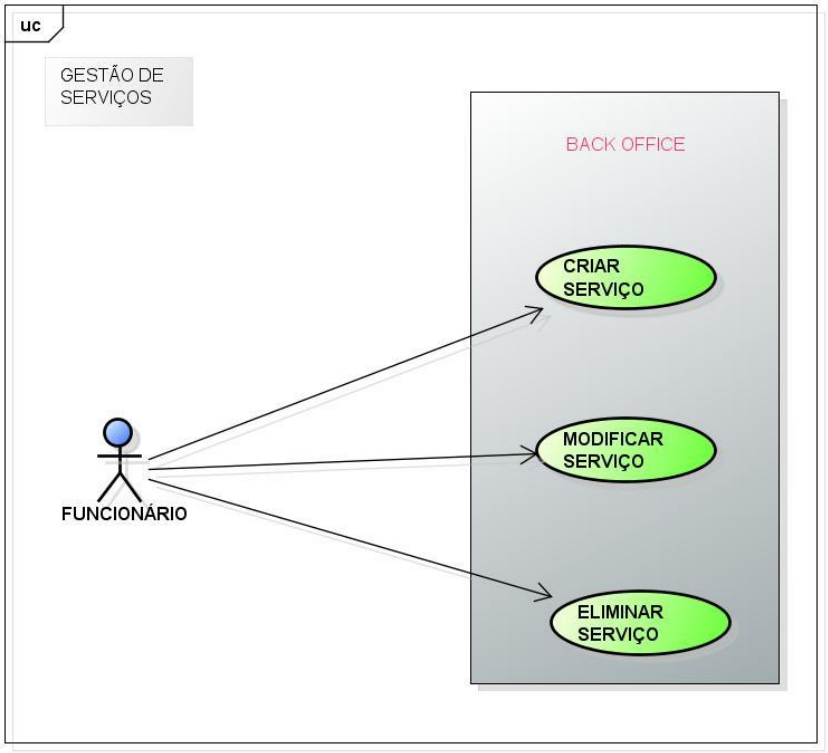


Figura 15 - Caso de Uso Gestão de Serviços

Casos de uso – Gestão de balcões

Assim como na gestão de serviços apenas funcionários com permissões de administrador podem efetuar as operações. Tal como mostra figura 16, as operações são: criar, modificar e eliminar balcões de atendimento.

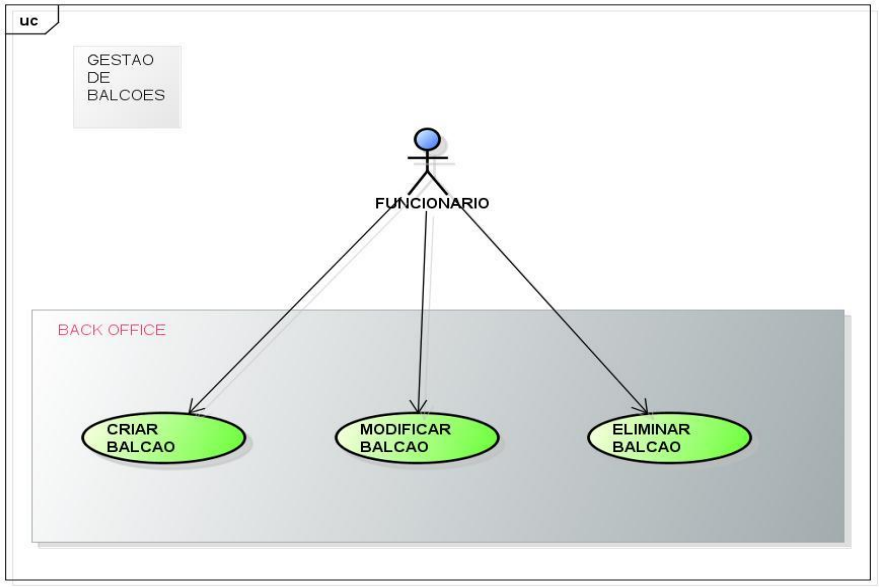


Figura 16 - Caso de Uso Gestão de Balcões

Casos de uso – gestão de funcionários

Todos os funcionários da empresa receberão privilégios que os permitam entrar no sistema. Todos terão um *username* e uma *password* para entrarem no sistema. Os funcionários com privilégio de administrador poderão criar, modificar ou eliminar um determinado funcionário, como mostra a figura 17. Os outros funcionários poderão efetuar o *login* e *logout* num chamador.

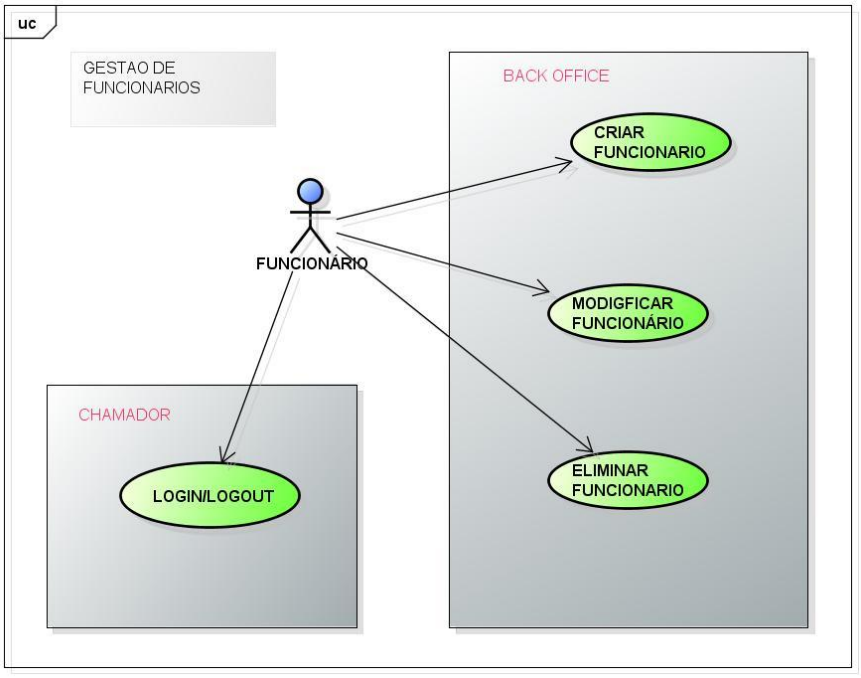


Figura 17 - Caso de Uso Gestão de Funcionários

Casos de uso - consultar estado de senha e de serviço

Ao escolher o serviço, o cliente pode observar diretamente o número de pessoas que estão na fila para um determinado serviço e ainda pode visualizar no *display* a última senha que foi chamada e a que está sendo atendida.

Na senha impressa estarão especificados o número de pessoas que estão na fila, o tempo médio de atendimento para cada cliente e ainda outras informações como hora, data, entre outros.

Como mostra a figura 18 também o funcionário pode consultar o estado de serviço, porque o seu *interface* (chamador) vai apresentar todas as informações do seu serviço tais como o número de pessoas na fila, tempo médio de atendimento, número de senha atendidas, entre outros.

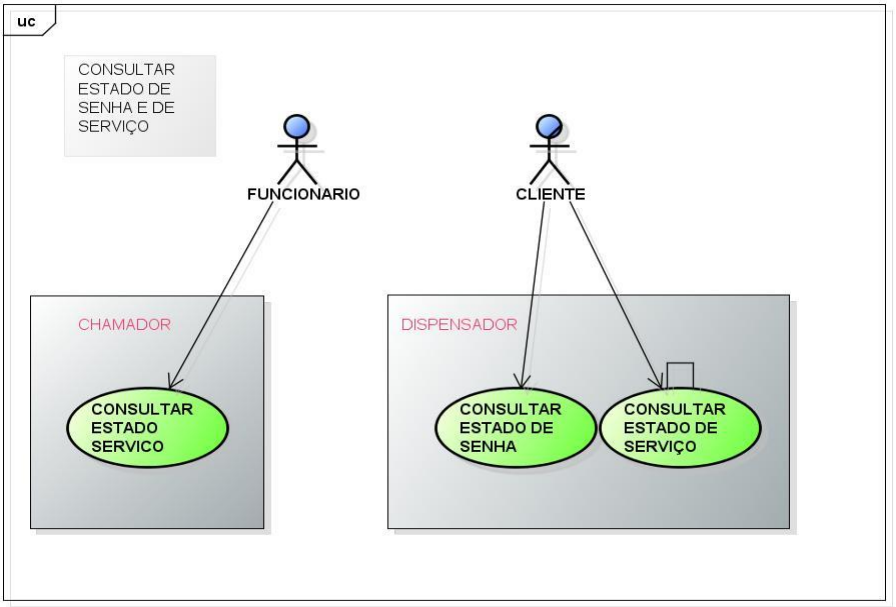


Figura 18 - Caso de Uso consultar estado de senhas e de serviços

Casos de uso – gestão de SMS

Ao pedir a senha para um determinado serviço o cliente escolhe a opção que lhe permite receber alerta por SMS.

Para que tenha o seu número de telemóvel validado, receberá um SMS contendo o código que lhe dará acesso ao serviço. O código será pedido sempre que o cliente requisitar o serviço. O SMS de envio de código não será cobrado, mas o SMS de alerta será cobrado assim que a senha for levantada, caso o cliente não tenha créditos no telemóvel não receberá a mensagem.

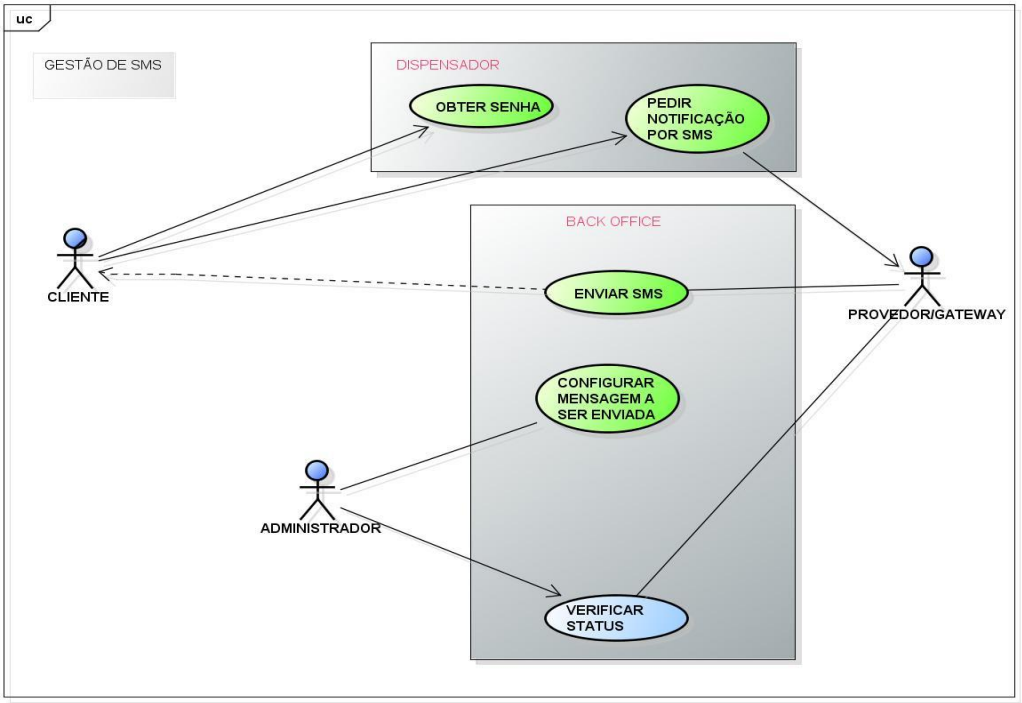


Figura 19 - Caso de Uso Gestão de SMS

Casos de uso – Visualizar conteúdos multimédia

O cliente que deseja esperar na fila pode visualizar conteúdos multimédia previamente configurados. Como mostra a figura 20 esses conteúdos são visualizados num *display* fixo.

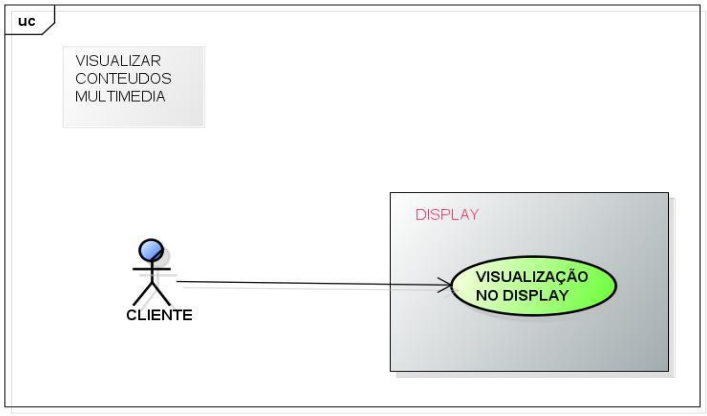


Figura 20 - Caso de uso visualizar conteúdos multimédia

3.3.3. Diagramas de Sequência

Um diagrama de sequência mostra uma interação, que representa a sequência de mensagens entre instâncias de classes, componentes, subsistemas ou atores. O tempo flui para baixo no diagrama e mostra o fluxo de controlo de um participante para outro.

De seguida apresentarei os diagramas de sequência mais significativos de algumas ações do meu sistema:

Obter senha

As senhas são obtidas no dispensador, onde o cliente escolhe o serviço e as demais opções disponíveis. As informações são enviadas para o servidor responsável por gerar a senha, guardadas na base de dados e em seguida importadas para o *display* onde serão apresentadas, como se pode observar na figura 21. Finalmente, a senha é impressa no dispensador.

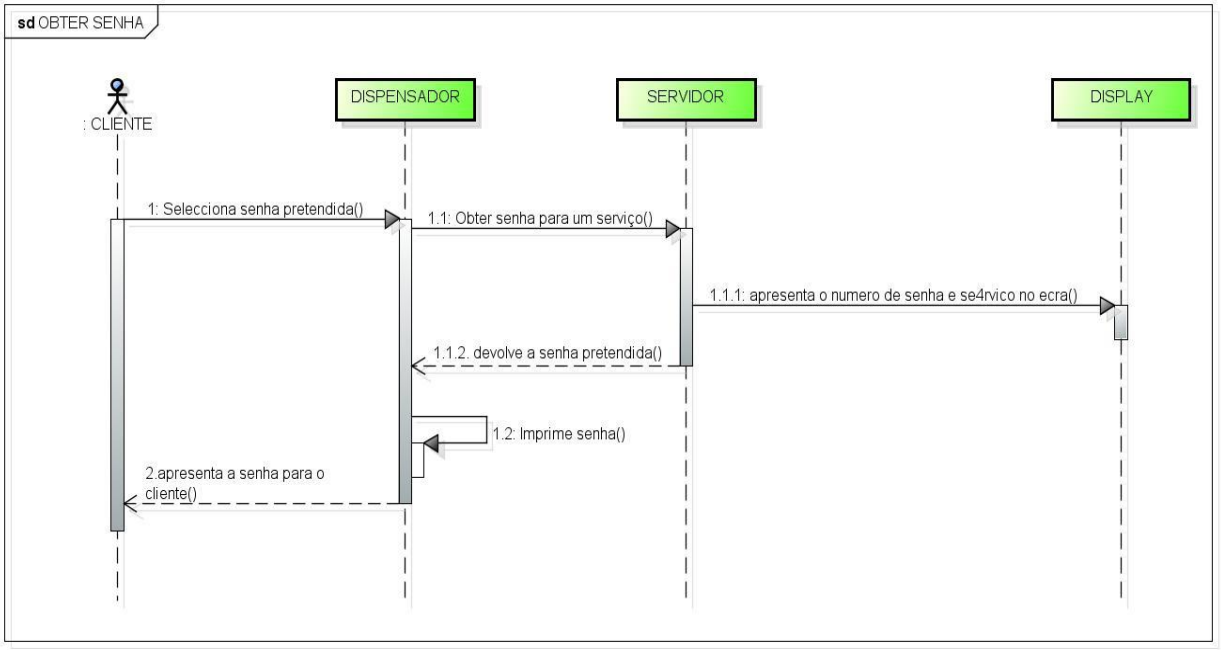


Figura 21 - Diagrama de Sequencia-obter senha

Obter senha com alerta SMS

A figura 22 descreve o diagrama de sequência ‘ Obter senha com alerta SMS’, onde o cliente solicita os serviços disponíveis que são automaticamente apresentados no ecrã; pode então solicitar a sua senha, escolher o seu serviço e escolher a notificação por SMS, onde vai ser exigido que o cliente digite o seu número de telemóvel para que possa posteriormente receber o alerta.

As informações são enviadas para o servidor e guardadas na base de dados. Caso o cliente possua crédito, o servidor envia o pedido ao provedor de SMS que irá providenciar o envio da mensagem. A mensagem chegará até o cliente através do *Gateway* SMS que é o fornecedor de créditos SMS ao provedor.

Após o envio de mensagem, o provedor de SMS receberá um *status* de envio, que por sua vez vai notificar o servidor do sistema sobre o sucesso do envio da mensagem.

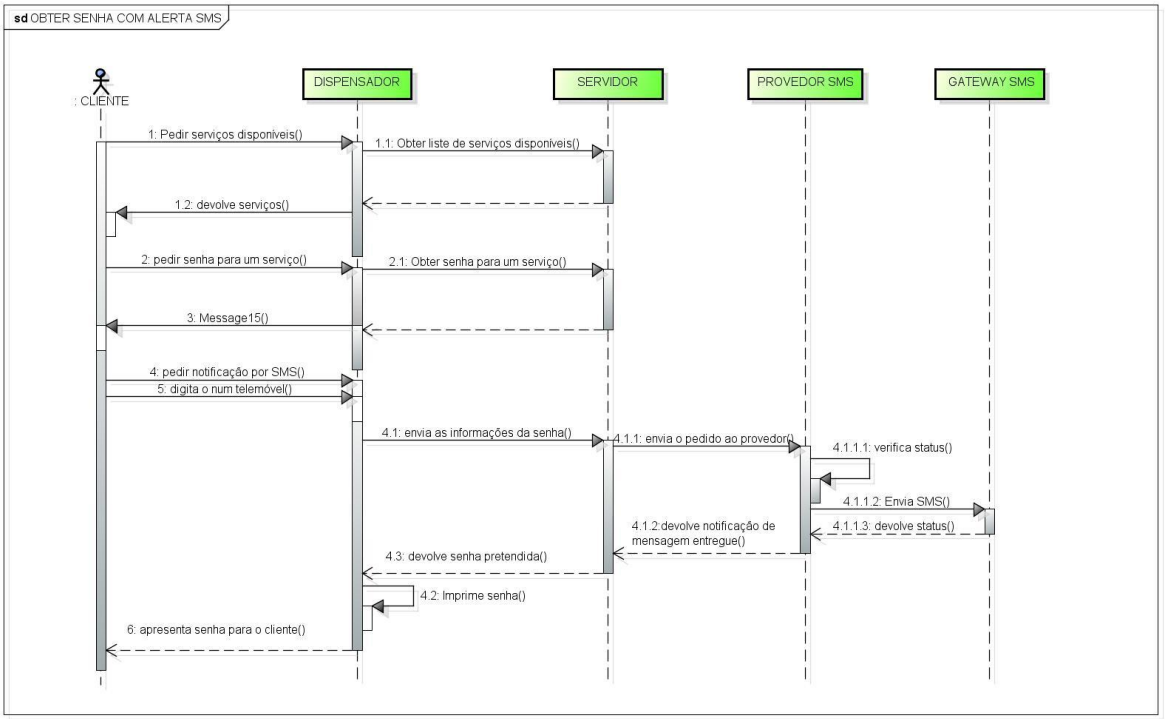


Figura 22 - Diagrama de sequência - obter senha com alerta SMS

Chamar senha

Como se pode observar na figura 23 o funcionário chama a próxima senha de um dado serviço. Feito isto, o servidor devolve a próxima senha ao chamador, guarda as informações na base de dados e envia as informações para o *display* de visualização.

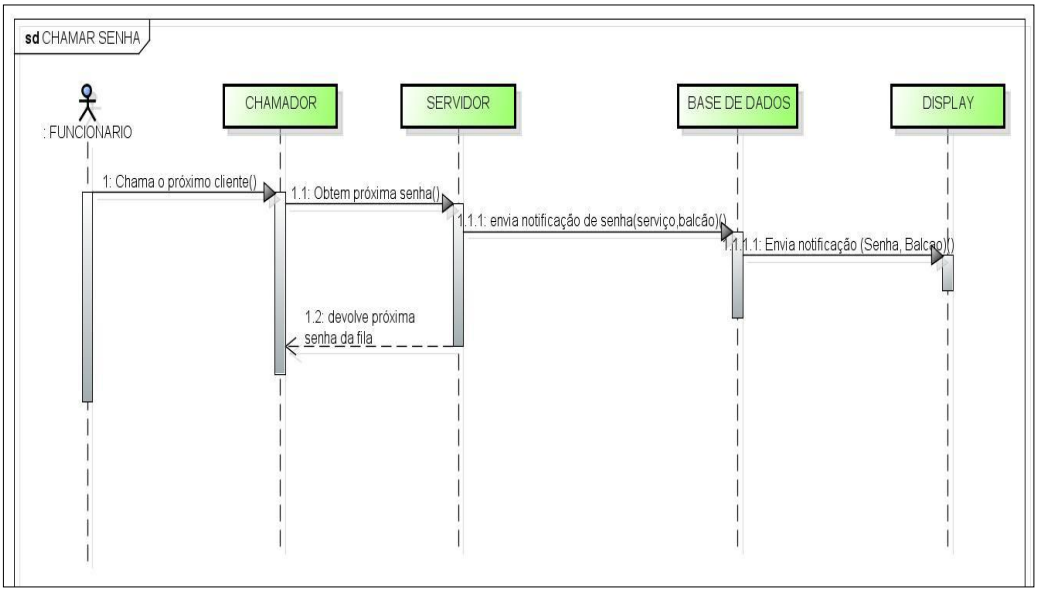


Figura 23 - Diagrama de sequencias - Chamar senha

3.4. Modelo De Dados

Modelar significa criar um modelo que explique as características de funcionamento e comportamento de um *software* a partir do qual ele será criado, facilitando o seu entendimento e projeto, através das características principais que evitarão erros de programação, projeto e funcionamento. É uma parte importante do desenho de um sistema de informação.

A figura 24 mostra o esquema da base de Dados através do diagrama do modelo EER, desenhada na ferramenta MySQL Workbench.

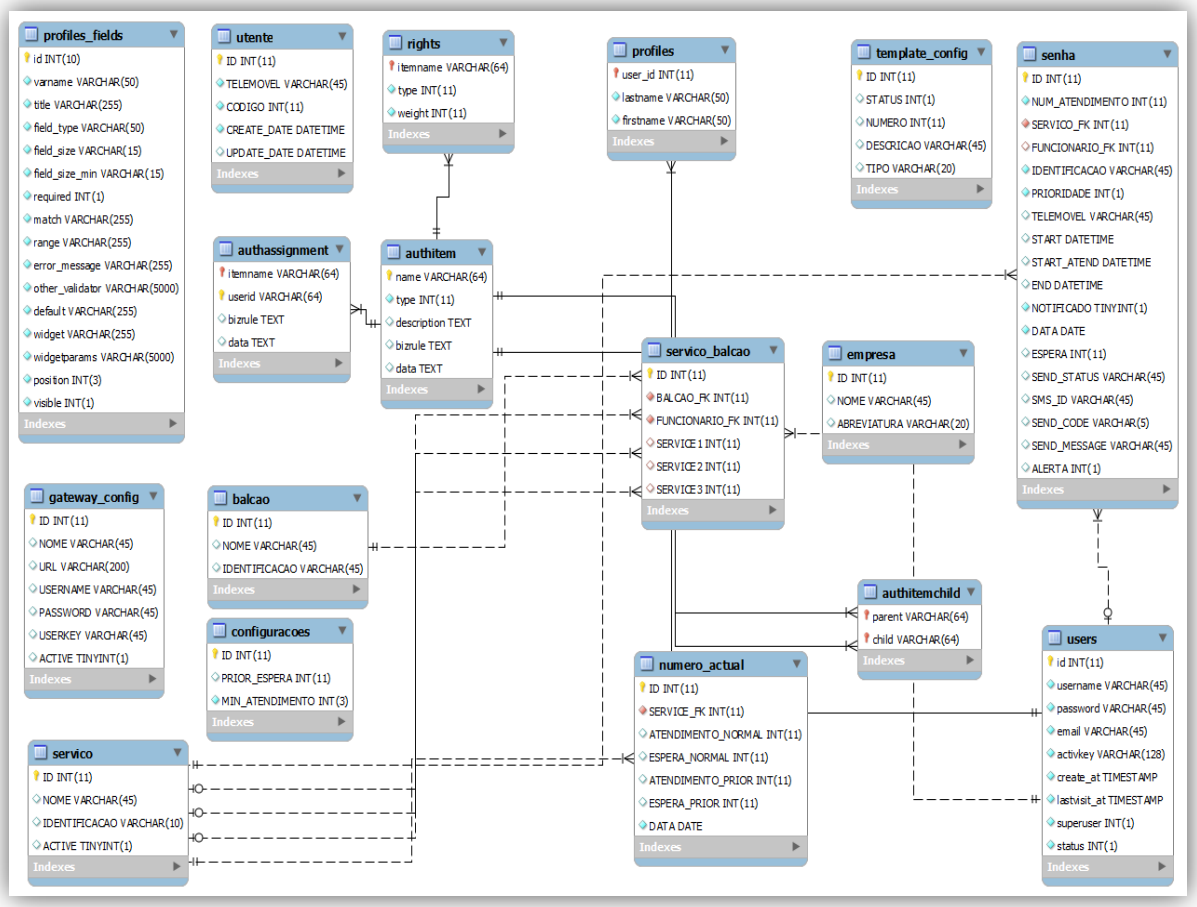


Figura 24 - Diagrama do modelo ER

UTENTE – entidade que define um cliente, utilizador do sistema

Tabela 1 - Descrição da entidade utente

Nome do atributo	Descrição
Id	Identificador único de cliente
Telemóvel	Número de telemóvel do cliente
Código	Código de autenticação do cliente
Create_date	Data de utilização
Update_date	Atualizar data

USERS – entidade que define a conta de usuário

Tabela 2 - Descrição da entidade Usuário

Nome do atributo	Descrição
Id	Identificador único de funcionário
Username	Nome de utilizador de um funcionário
Password	Palavra passe de acesso de um funcionário
Email	Endereço de *email* de um funcionário
ActivKey	Chave de ativação
Lastvisit_at	Guarda a informação de última visita
Superuser	Indica se é super utilizador ou utilizador normal

SERVICE_BALCAO – Entidade que define a relação de cliente com balcão

Tabela 3 - Descrição da entidade Serviço_Balcao

Nome do atributo	Descrição
Id	Identificador único de um serviço de um balcão
Balcão_fk	Balcão como chave estrangeira
Funcionário_fk	Funcionário como chave estrangeira

BALCAO – Entidade que define um determinado balcão

Tabela 4 - Descrição da entidade Balcão

Nome do atributo	Descrição
Id	Identificado único de balcão
Nome	Nome do balcão
Identificação	Número de identificação do balcão

EMPRESA – Entidade que define uma empresa que adquire o sistema

Tabela 5 - Descrição da entidade Empresa

Nome do atributo	Descrição
Id	Identificador único da empresa
Nome	Nome da Empresa
Abreviatura	Abreviatura da empresa

SENHA – Entidade que define a senha

Tabela 6 - Descrição da entidade Senha

Nome do atributo	Descrição
Id	Identificador único da senha
Num_atendimento	Número que identifica o atendimento
Servico_fk	Serviço requisitado na senha
Funcionario_fk	Funcionário relacionados ao serviço
Identificacao	Identificação de senha com a sigla de serviço e num balcão
Telemóvel	Número de telemóvel do cliente que receberá a alerta
Start	Início do processo de requisição
End	Fim do processo de requisição
Notificado	Se vai ser ou não notificado
Data	Data de senha
Espera	Tempo de espera
SMS_ID	Identificação única do SMS requisitado
Send_Code	Código de autenticação cliente
Send_message	Mensagem de notificação com código

SERVICE – Entidade que define o serviço

Tabela 7 - Descrição da entidade Serviço

Nome do atributo	Descrição
Id	Identificador único de serviço
Nome	Nome do serviço
Abreviatura	Letra que abrevia o serviço

Active	Indica estado do serviço

GATEWAY_CONFIG – Entidade que define a configuração de mensagens

Tabela 8 - Descrição da entidade Gateway_config

Nome do atributo	Descrição
Id	Identificador único de configuração de SMS
Nome	Nome do provedor
URL	Caminho de acesso do sistema ao *gateway*
Username	Nome de utilizador
Password	Palavra passe de acesso ao módulo SMS
Userkey	Chave de acesso
Template	Tipo de *templates* mensagem
Active	Estado

NÚMERO_ACTUAL – Entidade que define a senha que está a ser atendida.

Tabela 9 - Descrição da entidade Numero_actual

Nome do atributo	Descrição
Id	Identificado único de número atual
*Service*_fk	Chave estrangeira de serviço
Atendimento_Normal	Número de pessoas em atendimento normal
Espera_Normal	Número de pessoas em atendimento prioritário
Atendimento_prior	Número de pessoas em espera normal
Espera_prior	Número de pessoas em espera prioritária

Date	Corrente data de atendimento

CONFIGURAÇÕES – Entidade que define as configurações do sistema

Tabela 10 - Descrição da entidade Configurações

Nome do atributo	Descrição
Id	Identificado único de Configurações
Prior_espera	Número de pessoas em espera prioritária
Min_atendimento	Número mínimo de pessoas que podem ser atendidas antes dum atendimento normal

4. PROTÓTIPO DO SISTEMA

Neste capítulo mostrarei todos os aspetos que envolveram a implementação do sistema desde o desenho da base de dados até a implementação do especto gráfico e o desenho das janelas. É importante referir-se que o sistema não foi implementado na sua totalidade, mas criou-se este protótipo com o objetivo de aproximar o mais possível de um sistema real, ficando por concluir algumas funcionalidades, que serão referidas como trabalho futuro.

A implementação teve início na análise do desenho mostrado na figura 25, onde foi definido a linguagem de programação a ser utilizada em cada componente do sistema, bem como o servidor e base de dados.

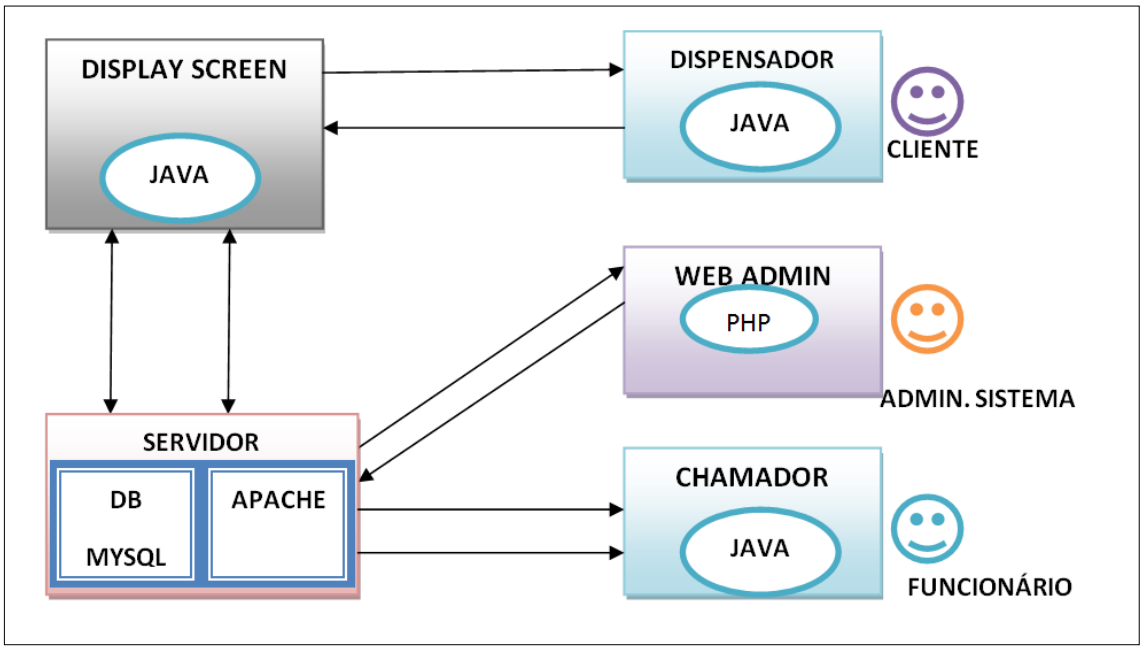


Figura 25 - Arquitetura do sistema

4.1. Implementação Base de Dados

A base de dados foi projetada partindo-se do desenho; começou-se por definir quais seriam as entidades do sistema e que relações teriam entre si. Como ferramenta utilizou-se o *Workbench*, que é uma ferramenta *MySQL*, de extrema importância e facilidade,

proporcionando a maneira mais fácil de desenhar uma base de dados e que depois de arquitetada foi diretamente importada para o SQl.

4.1.1. Descrição dos procedimentos MySQL

Estes procedimentos foram criados diretamente na base de dados com o objetivo de melhorar o desempenho e rapidez no acesso aos dados e na execução dos processos. A tabela 11 mostra uma descrição dos procedimentos mais importantes.

Tabela 11 - Descrição dos procedimentos MySQL

Procedimento	Descrição
SELECT UCASE(letra) into letra; SELECT ID INTO idService FROM servico WHERE IDENTIFICACAO = letra LIMIT 1;	Procura o id do serviço da letra
SELECT DATA INTO tbDate FROM numero_actual WHERE SERVICE_FK = idService; if tbDate != CURDATE() THEN	Verifica se os valores atuais da tabela número_actual são deste dia
UPDATE numero_actual set DATA = CURDATE (), ESPERA_NORMAL='0', ESPERA_PRIOR='0', ATENDIMENTO_NORMAL='0', ATENDIMENTO_PRIOR='0' WHERE SERVICE_FK=idService; END IF;	Verifica se os valores atuais da tabela número_actual são deste dia
UPDATE numero_actual set DATA = CURDATE (), ESPERA_NORMAL='0', ESPERA_PRIOR='0', ATENDIMENTO_NORMAL='0', ATENDIMENTO_PRIOR='0' WHERE SERVICE_FK=idService; END IF;	Se não faz a atualização da tabela.
if (idService is not null) then	Continuar apenas se o serviço existir

SELECT ESPERA_NORMAL INTO ultimoDaFila FROM numero_actual WHERE SERVICE_FK = idService limit 1;	Procura o valor atual da espera normal do serviço.
SELECT SUM (ultimoDaFila+1) INTO ultimoDaFila;	Adiciona 1 valor ao número atual do serviço
SELECT CONCAT (letra,ultimoDaFila) INTO identif;	Cria a identificação da senha pela concatenação da letra e numero.
UPDATE numero_actual SET ESPERA_NORMAL = ultimoDaFila WHERE SERVICE_FK = idService;	Atualiza o número atual do serviço.
INSERT INTO senha (NUM_ATENDIMENTO,SERVICO_FK,FUNCIONARIO_FK, IDENTIFICACAO, PRIORIDADE, ALERTA,TELEMOVEL, START, END, NOTIFICADO, DATA, ESPERA, SEND_STATUS, SMS_ID, SEND_CODE, SEND_MESSAGE) VALUE(ultimoDaFila, idService, null, identif, false, false, null, now(), null, false, curdate(), null, null, null, null, null);	Insere a senha na base de dados
select * from senha where NUM_ATENDIMENTO=ultimoDaFila and SERVICO_FK=idService and DATA = curdate();	Retorna a senha criada.
SELECT UCASE(letra) into letra; SELECT ID INTO idService FROM servico WHERE IDENTIFICACAO = letra LIMIT 1;	Procura o id do serviço da letra
SELECT SEC_TO_TIME(AVG(TIME_TO_SEC(TIMEDIFF(`END`,`START_ATEND`)))) AS TMA FROM senha WHERE FUNCIONARIO_FK IS NOT NULL AND END IS NOT NULL AND START IS NOT NULL AND START_ATEND IS NOT NULL AND YEAR(START) = YEAR(CURDATE())	Calcula a média da duração entre o *end* e o *start* da senha

AND DAYOFYEAR(START) = DAYOFYEAR(now()) AND TIME_TO_SEC(TIMEDIFF(`END`,`START_ATEND`))>=(SELECT MIN_ATENDIMENTO FROM configuracoes limit 1);	

4.2. Descrição dos Principais Métodos

Foram vários métodos utilizados para construção do sistema, mas serão destacadas na tabela abaixo os de maior importância.

Tabela 12 - Descrição dos principais métodos

Método	Descrição
public ConectionDB (String server, String dataBase, String user, String pass) {	Acesso à Base de Dados
public Response SendSMS send(Gateway_config config,int template, String contacto, String message) throws MalformedURLException, IOException {	Configuração do Envio de SMS

public Servico_Balcao servicesDoFuncionario(int funcionario) throws SQLException {
 	
return new Servico_Balcao_dao(conexao).getBalcaoService(funcionario);
	Colocar um funcionário num determinado balcão

public void notificarCliente(int service, int numActual) {	Notificação para o cliente
Gateway_config config = new Gateway_config_dao(conexao).myGateway();	Pega o *template* de envio de sms
Empresa emp = new Empresa_dao(conexao).myEmpresa();	Procura a empresa
List<Senha> lista = new Senha_dao(conexao).porNotificarAgora(service,	Pega todos os *tickets* que devem ser notificados nesse

numActual);	instante
public Senha novoTicketNotificacao(String letra, String pessoasEspera, String telemovel) {	Tirar uma senha com notificação
public Senha novoTicketPrioritario(String letra) {	Tirar uma senha com prioridade
public Senha novoTicketNormal (String letra) {	Tirar uma senha sem prioridade
public String getTMA_servico(int service) {	Retorna o TMA de um serviço
public Senha novoAtendimentoNormal(int service, int funcionario) {	Chamar uma nova senha
public int numeroPessoasAtendidasFunc(int funcionario, int service) {	Retorna o número de pessoas na fila
public int numeroPessoasAtendidasServico(int service) {	Retorna o número de pessoas atendidas por um determinado serviço

4.3. Implementação Do Dispensador

O dispensador de senhas foi o primeiro a ser implementado utilizando a linguagem java, sendo que as suas configurações iniciais iriam afetar todo o resto do sistema. Comecei por desenhar o formato do dispensador no *NetBeans*, com todos os botões e de seguida a programação dos seus eventos. Isso é uma das vantagens de utilização do *NetBeans*, por ser mais fácil conseguir o código depois do desenho, em vez de desenhar só por meio de código.

Partindo de um desenho básico de um dispensador, implementei o *interface* que permite inserir o número de telemóvel, um botão de serviço prioritário e um campo de texto que permite observar as informações de número de pessoas num determinado serviço.

A figura 26 mostra o *layout* do dispensador do sistema. Este display é adaptado para funcionar em ecrã *touch screan* de modo a proporcionar ao cliente uma melhor interação com o sistema

Figura 26 - Layout do Dispensador

O nome da empresa é configurado assim que é adquirido e todas as *interfaces* são adaptadas ao nível da empresa. No lado esquerdo contem a lista de serviços, configuráveis dependendo do número de serviços de cada empresa. O serviço prioritário é uma configuração fixa, podendo ser eliminada apenas pelo programador. Ao ser escolhido um serviço, o dispensador mostra diretamente o número de pessoas em espera naquele serviço, sendo que assim o cliente pode desistir sem ter que danificar uma senha.

Se o cliente optar por receber notificação via SMS, basta digitar o seu número de telemóvel e automaticamente é gerado um código que é enviado ao cliente, por via de SMS.

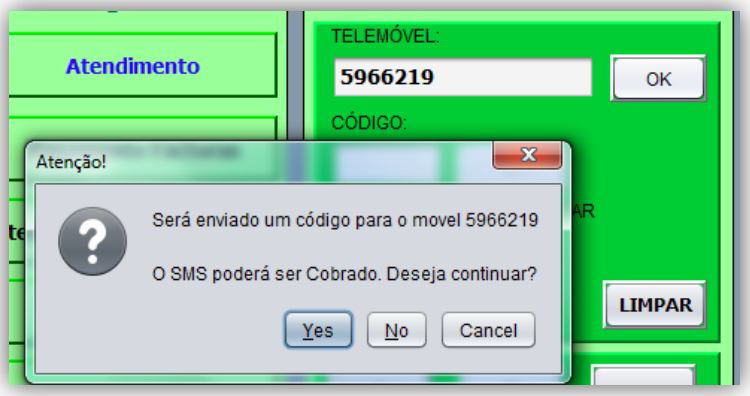


Figura 27 - Serviço com SMS

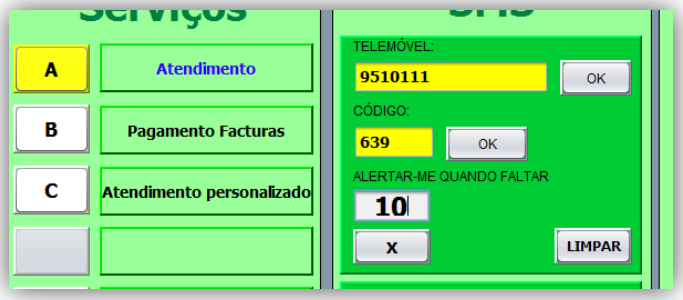


Figura 28 - Configurações necessárias para o serviço SMS

O cliente recebe uma mensagem contendo o código e uma mensagem dizendo para guardar o código para futuras utilizações no sistema. O cliente digita o código de autenticação e ainda pode escolher o número de pessoas que deverão estar à sua frente no momento em que lhe é enviado o alerta. Normalmente esse número já vem personalizado com o número 5. A partir do momento em que aperta o botão OK, o SMS ficará automaticamente configurado aguardando o momento do seu envio. De seguida é impressa uma senha com o seguinte formato:

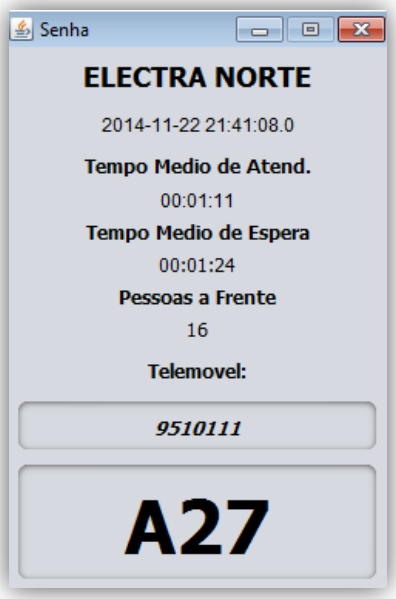


Figura 29 - Layout da senha

A senha contém a identificação da empresa, apresenta a data e a hora corrente do atendimento e ainda informações sobre o tempo médio de atendimento, tempo médio de espera, número de pessoas à frente, o número de telemóvel; caso for uma senha com notificação, em baixo mostra a identificação do serviço e número da senha.

Validação de número de telemóvel

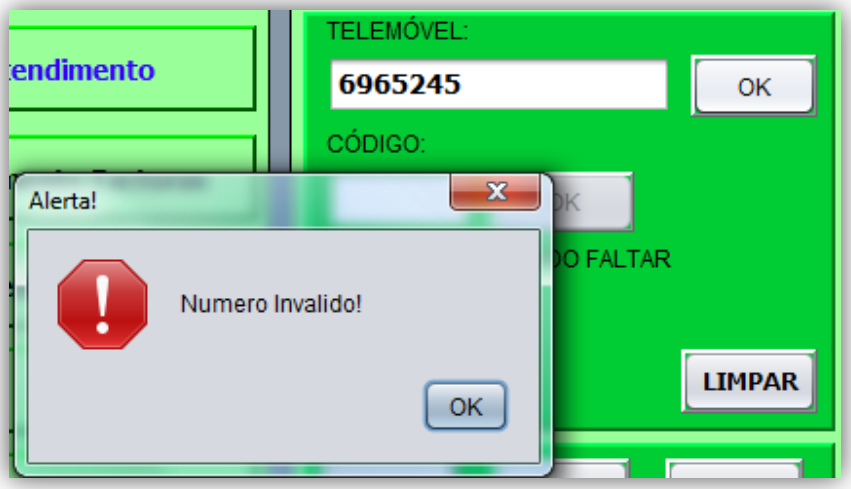


Figura 30 - Validação de número de telemóvel

O número de telemóvel tem que conter 7 dígitos, começar com os algarismos 5 ou 9, pois, caso contrário, o sistema dará um alerta de número inválido.

Senha com prioridade

Aplicável aos idosos, deficientes, gravidas, e acompanhantes de crianças ao colo. A senha prioritária segue uma regra, e é configurado no sistema. A senha prioritária também tem de esperar por sua vez. No sistema o serviço é configurado de forma que possa chamar 2 pessoas normais, 2 prioritários, e assim sucessivamente, de modo a evitar algum constrangimento, no caso de haver um número excessivo de pessoas com prioridade.

Figura 31 - serviço prioritário

4.4. Implementação Da conta de funcionário (Chamador)

Foi implementado em java e é por onde o funcionário chama as senhas para o atendimento, controla o estado do serviço e das senhas. Iniciou-se o seu desenho na ferramenta *NetBeans*, seguido da programação dos seus eventos. Foi a segunda parte do sistema a ser implementado.

A figura 32 mostra o *layout* da janela do chamador. Cada funcionário possui uma conta de utilizador, com *username* e *password*.

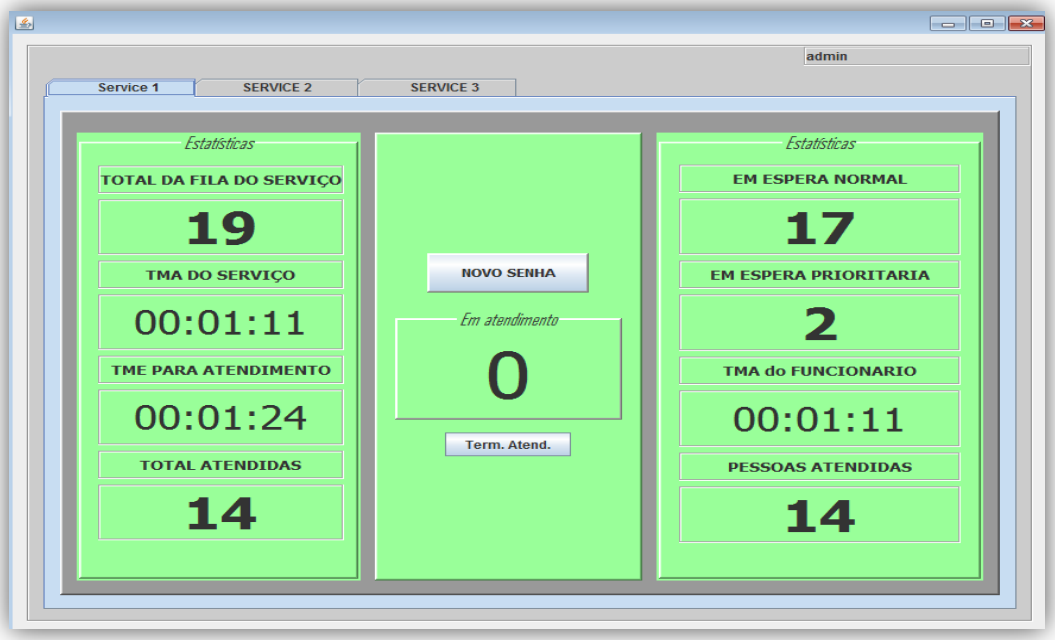


Figura 32 - Layout da interface conta de funcionário

O funcionário inicia o atendimento com os valores *ressetados*, e mostra o total das pessoas em espera do serviço. Mostra também a divisão entre pessoas em espera normal e em espera prioritária. Isso é importante para ter uma melhor noção do estado da fila. A partir do momento que chama um novo cliente, mostra a identificação da senha que está sendo atendida.

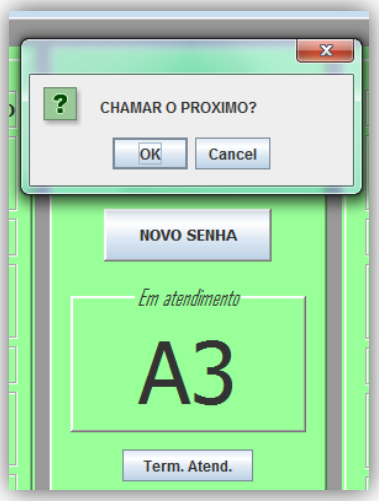


Figura 33 - Botão chamar nova senha

O botão *terminar atendimento* tem uma grande importância no cálculo do tempo médio de atendimento porque se não tiver mais ninguém na fila, a demora até o próximo atendimento não vai afetar no TMA.

É mostrado de forma separada, o TMA geral do serviço e o TMA, especifico do funcionário. Isso dá uma melhor noção do tempo ao funcionário. É possível também observar o número de pessoas atendidas.

4.5. Implementação Display

O *display* foi implementado em java, é a *interface* onde são apresentados as informações de serviço, senha e balcão, e mostra a última senha chamada. Para além disso, mostra também notas de rodapé, conteúdos multimédia e informações úteis, tais como podem ser observados nas figuras 34, 35 e 36.

A figura 34 mostra como pode-se aproveitar o Display para mostrar informações de estado do tempo na Cidade. De momento foi feito manualmente, mas pretende-se investigar um método que possa importar dados meteorológicos automaticamente, e estar sempre atualizados.

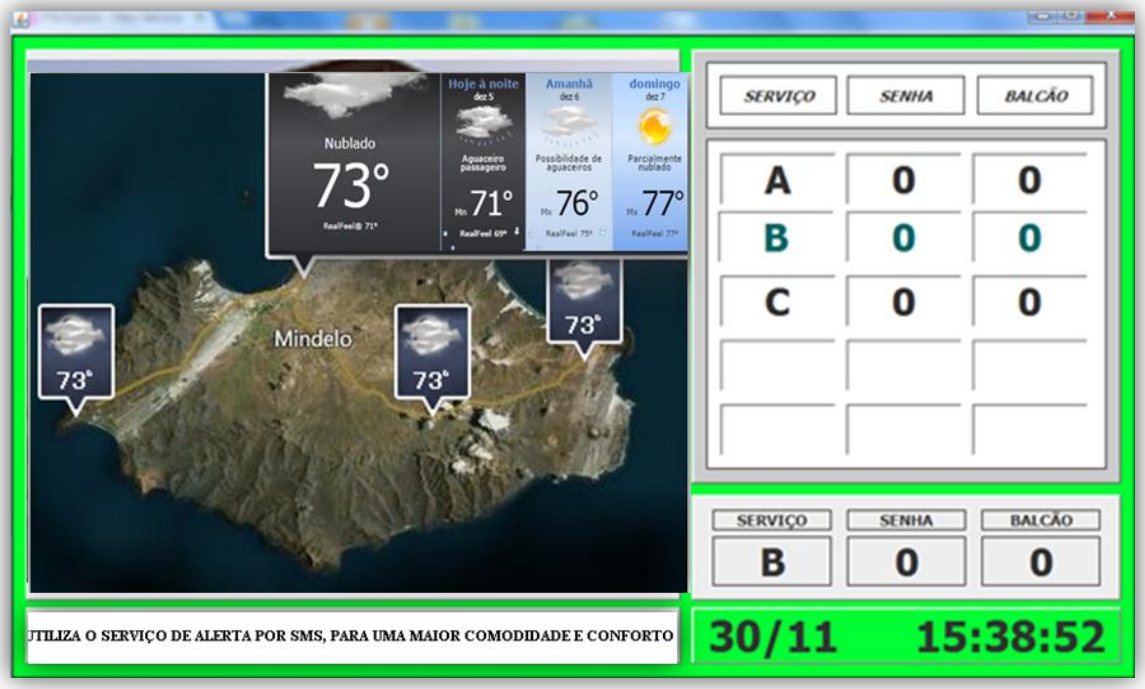


Figura 34 - *layout* display de apresentação

A figura 35, mostra que o display pode ser muito bem aproveitado para exibição de publicidades diversas, podendo ser da própria empresa, ou de empresas externas, e é um serviço que traz algum rendimento para a empresa, porque poderá cobrar uma taxa para exibir mensagens publicitárias.

Figura 35 - Mensagens publicitárias no display

O cliente quer estar sempre informado, e o display pode ser muito útil para transmitir informações que podem ser muito úteis para os clientes, tais como farmácias de serviço, informações úteis, notícias de última hora, e muito mais.

Figura 36 - Farmácias de serviço e informações úteis no display

Para gerir estes conteúdos, será implementado um gestor multimédia, de modo que os conteúdos podem ser programados, seguindo uma ordem para apresentação dos mesmos. A área de serviço é configurável de acordo com o número de serviços que uma determinada empresa tiver. Quando se chama um novo serviço, este aparece na janela em baixo com destaque.

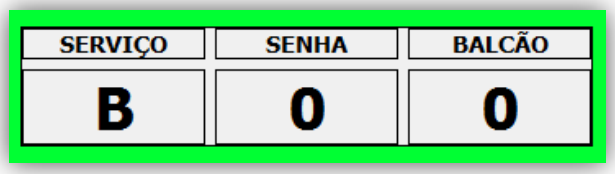


Figura 37 - apresentação da última senha chamada

As notas de rodapé passam mensagens e informações sobre a empresa, informações externas ou mensagens sobre o uso da fila de espera.

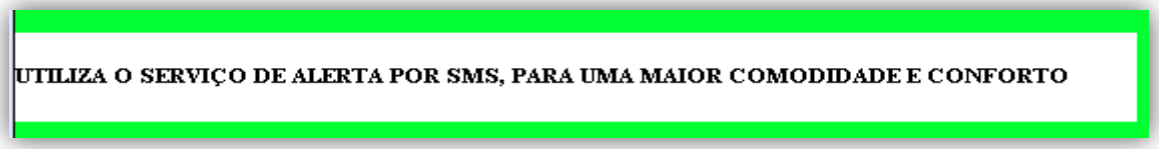


Figura 38 - Notas de rodapé

4.6. Implementação de *Interface Web Admin*

Essa *interface* foi implementada em *JavaScript* e linguagem PHP. É por onde a fila é gerida. Aqui são criados os funcionários, serviços, balcões e são definidas todas as regras de funcionamento do sistema. Através dessa interface pode-se atribuir um serviço à um determinado balcão, e o gestor pode colocar cada funcionário ao respetivo serviço, podendo sofrer mudanças a qualquer momento.

O logotipo foi uma criatividade do próprio autor, e procurou-se uma forma mais simples de representar uma fila. Para o fundo foi utilizado as ilhas de Cabo Verde, para mostrar que é um sistema de origem nacional.

A imagem abaixo mostra a página principal (*Home*), com uma nota de boas vindas.

Figura 39 - janela principal da interface Web admin

Ao clicar em *Login* serão requisitados o nome de utilizador e *password* para entrar no sistema.

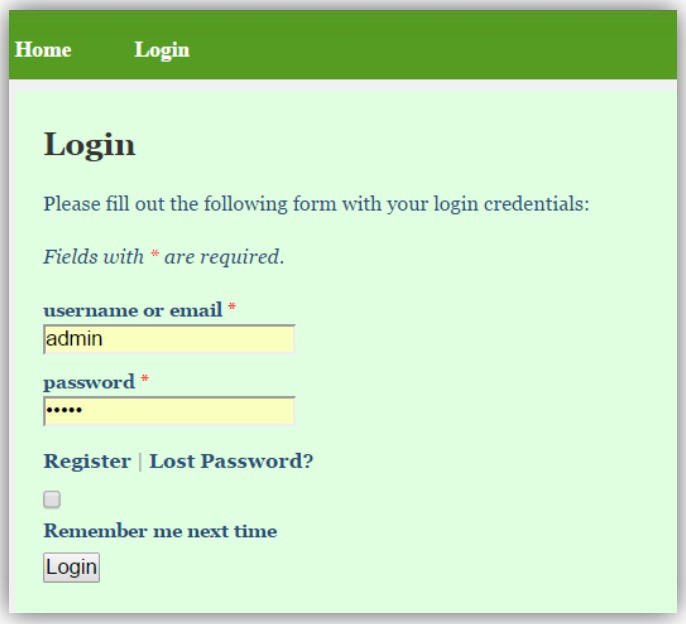


Figura 40 - Login Administrador

Pode acontecer que o utilizador do sistema esqueça o seu *password* de acesso, e para recuperação será pedido o seu email para que seja redirecionado uma mensagem de recuperação de *password*, tal como mostra a figura 41.

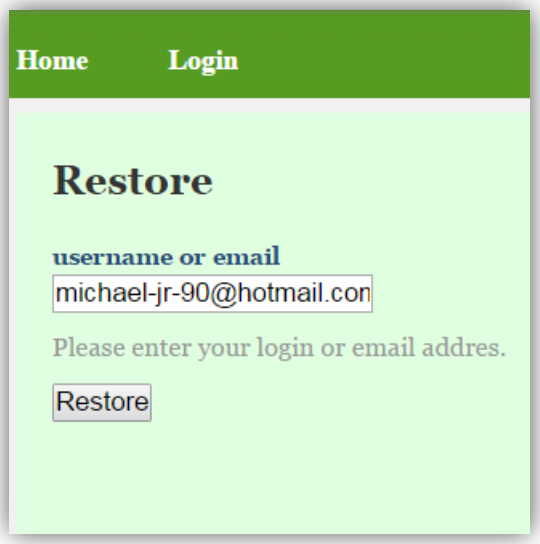


Figura 41 - Janela de recuperação de password

Após efetuar o *login*, aparece na página principal, os seguintes menus:

- Home
- Balcão
- Serviço
- Senhas
- Serviço & Balcão
- Logout

Figura 42 - apresentação dos Menus

Operações sobre balções

O menu balcão dá acesso à um conjunto de operações que se pode realizar sobre balcões, como mostra a figura 43, pode-se criar, listar, modificar e listar balcões.

Figura 43 - operações sobre balcões

A figura 44 mostra um exemplo de criação de balcão, e são listados um conjunto de balcões anteriormente criados. Os balcões são identificados por números em algarismos.

Figura 44 - Criando um novo balcão

A figura 45 mostra as operações que podem ser realizadas sob os serviços. Assim como os balcões, pode-se criar, listar, modificar e eliminar um determinado serviço. O sistema pode conter vários serviços, e estes serem ativados dependendo do tamanho e das necessidades da empresa.

Figura 45 - operações com serviços

A figura 46 mostra um exemplo de criação de um novo serviço.

Figura 46 - criando um novo serviço

A figura 47 mostra as operações que permitem controlar as senhas que já foram retiradas do sistema, e disponibiliza todos os dados que envolveram a retirada da senha. Pode-se

saber qual foi o serviço, qual foi o balcão, quem foi o funcionário que atendeu, hora e data, e outros.

Figura 47 - Operações com senhas

Figura 48 - analisando as senhas tiradas

O menu SERVIÇO & BALCÃO, atribui um serviço á um, ou mais balcões. Pode-se pegar num balcão anteriormente criado, atribuir um serviço, e colocar um funcionário no serviço. A figura 49 mostra essas operações.

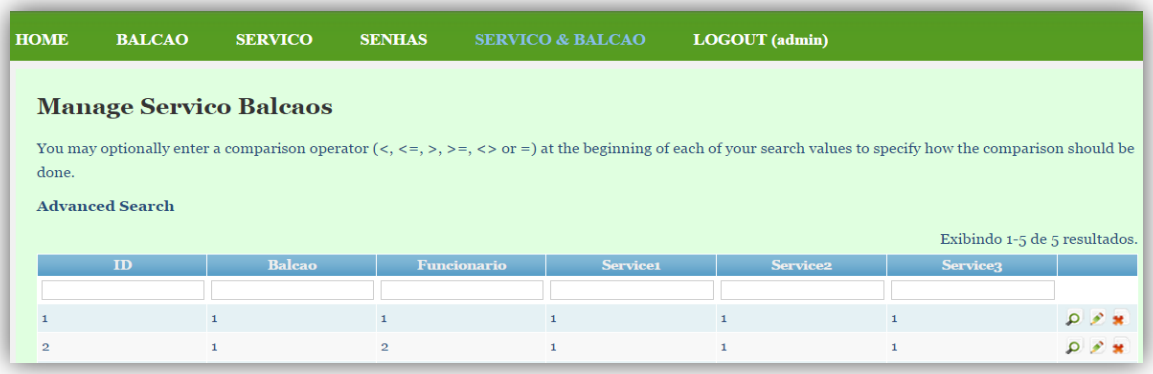


Figura 49 - Operações com Serviço - balcão

Para realizar qualquer operação neste menu, terão que existir serviços e balcões previamente criados para que se possa fazer a ligação entre os dois. A figura 50 mostra um exemplo de atribuição de um serviço à um determinado balcão, e introdução de funcionário no serviço. É importante destacar que, um serviço pode estar em vários balcões, mas o balcão é único, portanto só pode conter um serviço.

Figura 50 - criando um serviço com balcão e funcionário

5. CONCLUSÃO E RECOMENDAÇÕES

Para finalizar, serão apresentados neste capítulo as conclusões gerais do trabalho e também o trabalho futuro a realizar.

5.1. Conclusão do Trabalho

Procurou-se desenvolver um sistema que integra diversas tecnologias para proporcionar mais conforto e comodidade aos clientes. Esse sistema permite aos gestores das empresas gerir melhor a sua fila de espera, permite aos clientes gerirem melhor o seu tempo com utilização de alertas por SMS, bem como evitar que tenham que ficar fisicamente localizados num único lugar até que sejam atendidos.

A pesquisa realizada inicialmente sobre as filas de espera serviu de base para o arranque do projeto, impulsionando uma grande motivação para continuar, pois muitas vezes houve o risco de desistências em relação ao tema, mas na medida em que as dificuldades surgiam, as soluções sempre traziam algo inovador e mais ideias que permitiam o aperfeiçoamento cada vez mais do sistema.

Para realização desse projeto, procurou-se agregar as diversas tecnologias estudadas durante o curso, tais como programação (Java), Base de Dados (MySQL), análise de sistema (UML), programação Web (PHP) e outras que serviram de suporte para a implementação das diversas tecnologias.

Com o estudo realizado, percebeu-se a importância do estudo de tecnologias como forma de criar soluções que facilitem o dia-a-dia das pessoas e dos gestores das organizações, pois este Projeto facilita a vida dos clientes que utilizam as filas de espera, bem como os gestores que poderão gerir melhor a sua fila e analisar o desempenho dos seus funcionários, e com a introdução do sistema de alertas por SMS, faz com que este sistema se difere dos demais sistemas locais trazendo para as instituições públicas e privadas de Cabo Verde algo inovador para gerirem melhor as sua filas de espera.

Por fim pode-se dizer que os objetivos foram alcançados, as expectativas foram superadas e fica a certeza de que será um sistema aceite por qualquer entidade que deseja inovar e aproximar-se mais dos seus clientes.

5.2. Trabalho futuro a realizar

Quer-se deixar bem claro que o protótipo apresentado é apenas uma parte do que se pretendo para este sistema, pois quer-se aperfeiçoar uma versão comercial, pois há necessidade de um sistema mais completo nas instituições locais.

A solução apresentada é bastante completa, possui diversas funcionalidades que permitem uma gestão eficaz de uma fila de espera; contudo, ficaram por realizar alguns pontos necessários que permitam o funcionamento do sistema num cenário real.

Para uma versão futura, o SMS funcionará da forma em que o cliente pode, não só pedir para ser alertado ao aproximar a sua senha mas também pedir informações sobre o estado do sistema. De momento não foi possível fazer isso, porque isso exige que seja feita um *SMS Request*, ou seja, o cliente é que envia um SMS para receber outro como resposta ao pedido, e exige a programação para *modems*, e de momento o tempo era demasiado curto para aprofundar neste tema.

O dispensador será alterado para uma *interface* mais simples em que o cliente escolhe o serviço e tira a sua senha. A senha conterá todas as informações do sistema e um texto com uma mensagem que indicará ao cliente o número curto por onde enviará o SMS *request*, por exemplo:

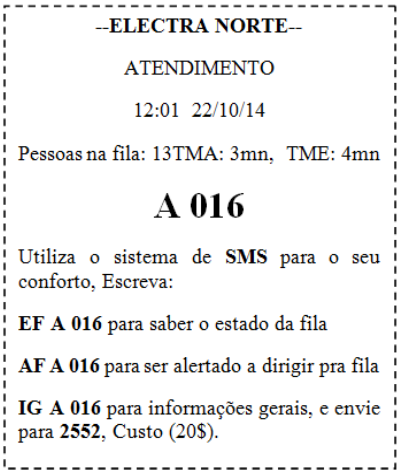


Figura 51 - proposta de configuração de senha

Sendo assim, a validação do cliente será feita directamente do seu próprio telemóvel e o SMS será cobrado, podendo ser a gestão de mensagens feita pela empresa, por uma operadora local ou por um provedor de Gateway SMS.

Sendo assim o cliente pode controlar melhor o seu tempo e a distância entre o local em que estiver com a área de atendimento.

O custo do SMS será definido de forma que seja dividido entre a empresa e a operadora, pois a empesa também tem que lucrar com o sistema; isto é mais um motivo para que o meu sistema seja aceite pelas instituições.

Para o sistema futuro, pensei também nos clientes que não vão utilizar o serviço SMS, ou seja, esses permanecerão na fila até serem atendidos. Terão à sua disposição um *display* para visualização de conteúdos, inclusive para publicidades, notícias da cidade, receitas de culinária, dicas do dia-a-dia, etc.

Há também a possibilidade de adaptar o este sistema para dispositivos *Androide,* tendo em conta o aumento do uso destes dispositivos. Sendo assim o cliente poderá aceder à sua senha diretamente no seu telemóvel, ou seja, não vai necessitar de senha em papel.

6. REFERÊNCIAS BIBLIOGRÁFICAS

Altronix, Gestão de Filas. **http://www.altronix.pt**, 2013-07-08, 12:53.

Análise dos requisitos funcionais e não funcionais, **http://homepages.dcc.ufmg.br/~figueiredo/disciplinas/aulas/req-funcional-rnf_v01.pdf**, 2014-01-20, 16:34.

António Miguel (2010). Gestão de Projetos de Software (4ª ed), Lisboa, FCA editora de informática, Lda.

Carlos Serrão & Joaquim Marques (2004). Programação com PHP 4.3, Lisboa, FCA editora de informática, Lda.

Cláudia Pereira (2009). Uma introdução às filas de espera. Dissertação de Mestrado.

Curso de java. **http://www.javaprogressivo.net/2012/08/curso-completo.html**, 2014-02-04, 17:32.

Engenharia de *software*. **http://www.devmedia.com.br/engenharia-de-software-2-tecnicas-para-levantamento-de-requisitos/9151**, 2013-12-18, 10:02.

F.Mario Martins (2001). Programação Orientada aos objetos em Java 2 (2ªed), Lisboa, FCA editora de informática, Lda.

Fernando Silveira (2008). Sistema Integrado de Gerenciamento de Senhas. Bacharelado, Centro Universitário de Brasília.

Ferramenta *Netbeans*. **https://netbeans.org/features/index_pt_BR.html**, 2014-05-05, 20:23.

Filas de espera, transparências de apoio nas aulas teóricas. **https://www.google.cv/url?sa=t&rct=j&q=&esrc=s&source=web&cd=7&cad=rja&uact=8&sqi=2&ved=0CDYQFjAG&url=http%3A%2F%2Fsigarra.up.pt%2Ffeup%2Fpt%2Fpubls_pesquisa.show_publ_file%3Fpct_gdoc_id%3D67577&ei=cwV-VLK-GMffaoqzgaAG&usg=AFQjCNH1zeYeJZdWZ_GsN-gHpGl6Tc3kyQ**. 2013-11-08, 14:02.

Gestão de atendimento e filas de espera Qmagine. **http://www.qmagine.com/store_pt/index_2010_v2.php?page=gestao_filas**, 2013-11-08.

Introdução à teoria das filas. **http://www.ceset.unicamp.br/~marlih/ST565/intro-filas.pdf**, 2013-07-08, 13:19.

Java orientação à objetos. **http://www.caelum.com.br/apostila-java-orientacao-objetos/**, 2014-02-04, 16:57.

José Luís Pereira (1998). Tecnologia de Bases de Dados (3ªed), Lisboa, FCA editora de informática, Lda.

Linguagem *Mysql*. **http://www.techtudo.com.br/artigos/noticia/2012/04/o-que-e-e-como-usar-o-mysql.html**, 2014-05-03, 15:25.

Linguagem *Netbeans*. **https://netbeans.org/index_pt_PT.html**, 2014-05-04, 20:01.

Linguagem PHP. **http://www.etelg.com.br/paginaete/downloads/informatica/php.pdf**, 2014-04-01, 16:23.

Linguagem PHP. **http://www.infoescola.com/informatica/php/**, 2014-04-01, 13:51.

Manual com truques e dicas HTML. **http://www.truquesedicas.com/tutoriais/html/00001b.htm**, 2014-05-06, 11:48.

Manual PHP. **https://php.net/manual/pt_BR/intro-whatis.php**, 2014-04-01, 16:18.

Mauro Nunes ed alii (2004). Fundamental de UML (4ª ed), Lisboa, FCA Editora de informática, Lda.

Programação e algoritmos. **https://eden.dei.uc.pt/~pa2/apresenta/paii_intro.pdf**, 2013-12-16, 19:32.

Programação em java. **http://www.tiexpert.net/programacao/java/**, 2014-02-04, 18:06.

Sistema de filas com informação via SMS. **http://www.infocontrol.pt/220/esirius.htm**, 2013-10-09, 20:03.

Sistema de filas *Newvision*. **http://www.newvision.pt**, 2013-11-08, 13:59.

SMS Gateway. **http://www.ceset.unicamp.br/~marlih/ST565/intro-filas.pdf**, 2013-10-09, 19:31.

Softwares livres. **https://www.gnu.org/philosophy/free-sw.pt-br.html**, 2014-01-16, 20:25.

Total gestão de filas e atendimento. **http://www.visioncast.net/pt/produtos/qnet.html**, 2013-10-14, 10:07.

Vítor carvalho (2012). Visualização de Sistemas de Atendimento e Corporate TV. Dissertação de Mestrado, universidade técnica de Lisboa.

Wikipédia, Teoria de Filas. **http://pt.wikipedia.org/wiki/Teoria_das_filas**, 2013-07-08,

ANEXOS

-----Código Fonte das principais funções----

[INÍCIO DA CLASSE LOGIN]

```
public class Login {

   public int fazLogin(String URL_WS, String username, String password) {

      int id = -1;
      if (URL_WS != null && username != null && password != null
            && !"".equals(URL_WS) && !"".equals(username) && !"".equals(password)) {

         try {
            URL_WS += "?username=" + username + "&password=" + password;
            // cria o objeto url
            URL url = new URL(URL_WS);
            // cria o objeto httpurlconnection
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            // conecta com a url destino
            connection.connect();
            // abre a conexão pra input
            BufferedReader input = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
            StringBuilder inputString = new StringBuilder();
            String s = "";
            //PERCORRE TODO O INPUT E CONCATENA AS LINHAS
            while (null != ((s = input.readLine()))) {
               inputString.append(s);
            }
            if (inputString.toString() != null && inputString.toString().matches("[0-9]")) {
               id = Integer.parseInt(inputString.toString());
            }
            input.close();
         } catch (MalformedURLException ex) {
            Logger.getLogger(Login.class.getName()).log(Level.SEVERE, null, ex);
         } catch (IOException ex) {
            Logger.getLogger(Login.class.getName()).log(Level.SEVERE, null, ex);
         }
      } else {
         System.err.println("Campo Vasio!");
      }
      return id;
   }

}
```

[FIM DA CLASSE LOGIN]

[INICIO DA CLASSE REPOSITÓRIO]

```
package application;

import application.sms.SendSMS;
import application.utils.InfoDisplay;
import application.utils.ResponseSendSMS;
```

```
import dao.*;
import java.awt.Image;
import java.awt.Toolkit;
import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import model.*;
import java.util.Date;
import javax.swing.JOptionPane;

public class Repositorio {

    Connection conexao;

    public Repositorio(Connection con) {
        this.conexao = con;
    }

    public Repositorio() {

    }

    public Image MEU_ICON() {
        return Toolkit.getDefaultToolkit().getImage(getClass().getResource("/image/logo.png"));
    }

    public Configuracoes myConfiguracao() throws SQLException {
        return new Configuracoes_dao(conexao).getConfiguracao();
    }

    public User Login(String user) {
        try {
            return new User_dao(conexao).getAccount(user);
        } catch (SQLException ex) {
            Logger.getLogger(Repositorio.class.getName()).log(Level.SEVERE, null, ex);
        }
        return null;
    }

    public Servico_Balcao servicesDoFuncionario(int funcionario) throws SQLException {
        return new Servico_Balcao_dao(conexao).getBalcaoService(funcionario);
    }

    public void notificarCliente(int service, int numActual) {
        try {           //PEGA AS CONFIGURACOES DE ENVIO DE SMS
            Gateway_config config = new Gateway_config_dao(conexao).myGateway();
            //PEGA O TEMPLATE DE ENVIO DE SMS
            Template_config template = new Template_config_dao(conexao).getTemplate("SMS");
            //PARA CONTINUAR DEVE VERIFICAR SE CONFIGURAÇAO E TEMPLATE NAO SAO
NULL
            if (config != null && template != null) {
                //buscar a empresa
                Empresa emp = new Empresa_dao(conexao).myEmpresa();
                //PEGA TODOS OS TICKETS QUE DEVEM SER NOTIFICADOS NESSE INSTANTE
                List<Senha> lista = new Senha_dao(conexao).porNotificarAgora(service, numActual);
```

```
            //ENVIA TODAS MENSAGENS, 1 POR 1 E ACTUALIZA CADA TICKET COMO
NOTTIFICADO
            for (Senha ticket : lista) {
                String message = "Alerta Fila Espera " + emp.getAbreviatura().toUpperCase() + ". Faltam " +
ticket.getEspera() + " pesssoas para seres atendido!";
                ResponseSendSMS response = new SendSMS().send(config, template.getNumero(),
ticket.getTelemovel(), message);
                if (response != null) {
                    //SETAR OS CAMPOS DE TIKET
                    ticket.setNotificado(true);
                    ticket.setSendStatus(response.getStatus());
                    ticket.setSmsId(response.getSMSid());
                    ticket.setSendCode(response.getCode());
                    ticket.setSendMessage(response.getMessage());
                    //ACTUALIZAR O TIKET
                    new Senha_dao(conexao).actualizar(ticket);
                } else {
                    System.err.println("Nao foi possivel enviar alerta para tiket " + ticket.getIdentificacao());
                }
            }
        } else {
            System.err.println("Gateway  ou template invalido!");
        }
    } catch (SQLException | IOException ex) {
        Logger.getLogger(Repositorio.class.getName()).log(Level.SEVERE, null, ex);
    }
}

public Senha novoTicketNotificacao(String letra, String pessoasEspera, String telemovel) {
    if (letra != null && telemovel != null && pessoasEspera != null) {
        try {
            return new Senha_dao(conexao).criarSenhaNotificacao(letra, telemovel, pessoasEspera);
        } catch (SQLException ex) {
            Logger.getLogger(Repositorio.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    return null;
}

public Senha novoTicketPrioritario(String letra) {
    if (letra != null) {
        try {
            return new Senha_dao(conexao).criarSenhaPrioritario(letra);
        } catch (SQLException ex) {
            Logger.getLogger(Repositorio.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    return null;
}

public Senha novoTicketNormal(String letra) {
    if (letra != null) {
        try {
            return new Senha_dao(conexao).criarSenhaNormal(letra);
        } catch (SQLException ex) {
            Logger.getLogger(Repositorio.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    return null;
```

```
}

public String getTMA_servico(int service) {
   return new Senha_dao(conexao).getTMA_SERVICO(service);
}

public String getTMA_funcionario(int funcionario, int service) {
   return new Senha_dao(conexao).getTMA_FUNCIONARIO(funcionario, service);
}

public String getTME_servico(int service) {
   return new Senha_dao(conexao).getTME_SERVICO(service);
}

public void fimDoAtendimento(int funcionario) {
   new Senha_dao(conexao).fimAtendimento(funcionario);
}

public NumeroActual numeroActualDoServ(int service) {
   try {
      return new NumeroActual_dao(conexao).getNumeroActual(service);
   } catch (SQLException ex) {
      Logger.getLogger(Repositorio.class.getName()).log(Level.SEVERE, null, ex);
   }
   return null;
}

public Servico getService(int id) throws SQLException {
   return new Servico_dao(conexao).getServiceID(id);
}

public Servico getService(String letra) throws SQLException {
   return new Servico_dao(conexao).getServiceIdentif(letra);
}

public List<Servico> todosService(){
   try {
      return new Servico_dao(conexao).allServices();
   } catch (SQLException ex) {
      Logger.getLogger(Repositorio.class.getName()).log(Level.SEVERE, null, ex);
   }
   return null;
}

public Empresa getEmpresa() throws SQLException {
   return new Empresa_dao(conexao).myEmpresa();
}

public void terminarAtendimento(int funcionario){
   new Senha_dao(conexao).fimAtendimento(funcionario);
}

public Senha novoAtendimentoNormal(int service, int funcionario) {
   try {
      return new Senha_dao(conexao).novoAtendimentoNormal(service, funcionario);
   } catch (SQLException ex) {
      Logger.getLogger(Repositorio.class.getName()).log(Level.SEVERE, null, ex);
   }
   return null;
```

```
    }

    public Senha novoAtendimentoPrior(int service, int funcionario) {
        try {
            return new Senha_dao(conexao).novoAtendimentoPrior(service, funcionario);
        } catch (SQLException ex) {
            Logger.getLogger(Repositorio.class.getName()).log(Level.SEVERE, null, ex);
        }
        return null;
    }

    public int numeroPessoasAtendidasFunc(int funcionario, int service) {
        try {
            return new Senha_dao(conexao).pessoasAtendidasFunc(funcionario, service);
        } catch (SQLException ex) {
            Logger.getLogger(Repositorio.class.getName()).log(Level.SEVERE, null, ex);
        }
        return 0;
    }

    public int numeroPessoasAtendidasServico(int service) {
        return new Senha_dao(conexao).pessoasAtendidasServico(service);
    }

    public Utente getSmsCliente(String telemovel) {
        try {
            return new Utente_dao(conexao).getUtente(telemovel);
        } catch (SQLException ex) {
            Logger.getLogger(Repositorio.class.getName()).log(Level.SEVERE, null, ex);
        }
        return null;
    }

    public String gerarCodigo() {
        String valor = String.valueOf(new Date().getTime());
        return valor.substring(valor.length() - 3, valor.length());
    }

    public void enviarSMSautenticacao(String telemovel, String codigo, String empresa) throws
SQLException, IOException {

        //cria a mensagem a ser enviada
        String mensagem = empresa.toUpperCase() + ".Codigo de autenticacao: " + codigo + ". Por favor
guarde o codigo para futuro utilizacao no sistema.";
        //pega as info de conexao com o gateway
        Gateway_config gcf = new Gateway_config_dao(conexao).myGateway();
        //pega otemplate de SMS
        Template_config temp = new Template_config_dao(conexao).getTemplate("SMS");
        //envia a mensagem
        if (gcf != null && temp != null) {
            new SendSMS().send(gcf, temp.getNumero(), telemovel, mensagem);
        } else {
            JOptionPane.showMessageDialog(null, "     O sistema nao processou o envio de SMS.\nProblema ao
identificar gateway de sms ou template utilizado!", "Erro!", 0);
        }
    }

    public int novoUtento(Utente utente) throws SQLException {
        return new Utente_dao(conexao).create(utente);
```

```
    }

    public List<InfoDisplay> ultimaSenhaCadaServico(){
        return new Senha_dao(conexao).ultimaSenhaCadaServico();
    }

}
```

[FIM DA CLASSE REPOSITÓRIO]

[INICIO DA CLASSE INFORMAÇÃO DE SENHA]

```
public class InformacaoSenha {
    private String Identificacao;
    private String Telemovel;
    private String Empresa;
    private String PessoasFrente;
    private String Hora;
    private String Data;
    private String tempMedAtend;
    private String tempMedEspera;

    public String getIdentificacao() {
        return Identificacao;
    }

    public void setIdentificacao(String Identificacao) {
        this.Identificacao = Identificacao;
    }

    public String getTelemovel() {
        return Telemovel;
    }

    public void setTelemovel(String Telemovel) {
        this.Telemovel = Telemovel;
    }

    public String getEmpresa() {
        return Empresa;
    }

    public void setEmpresa(String Empresa) {
        this.Empresa = Empresa;
    }

    public String getPessoasFrente() {
        return PessoasFrente;
    }

    public void setPessoasFrente(String PessoasFrente) {
        this.PessoasFrente = PessoasFrente;
    }

    public String getHora() {
        return Hora;
    }
```

```
    public void setHora(String Hora) {
        this.Hora = Hora;
    }

    public String getData() {
        return Data;
    }

    public void setData(String Data) {
        this.Data = Data;
    }

    public String getTempMedAtend() {
        return tempMedAtend;
    }

    public void setTempMedAtend(String tempMedAtend) {
        this.tempMedAtend = tempMedAtend;
    }

    public String getTempMedEspera() {
        return tempMedEspera;
    }

    public void setTempMedEspera(String tempMedEspera) {
        this.tempMedEspera = tempMedEspera;
    }

}
```

[FIM DA CLASSE INFORMAÇÃO DE SENHA]

[INICIO DA CLASSE SENHA]

```
public class Senha {

    private int ID;
    private int NumAtendimanto;
    private int Service;
    private int Funcionario;
    private String Identificacao;
    private boolean Prioridade;
    private String Telemovel;
    private String Start;
    private String End;
    private boolean Notificado;
    private String Date;
    private int  Espera;
    private String SendStatus;
    private String SmsId;
    private String SendCode;
    private String SendMessage;
    private boolean Alerta;

    public int getID() {
        return ID;
    }
```

```
public void setID(int ID) {
    this.ID = ID;
}

public int getNumAtendimanto() {
    return NumAtendimanto;
}

public void setNumAtendimanto(int NumAtendimanto) {
    this.NumAtendimanto = NumAtendimanto;
}

public int getService() {
    return Service;
}

public void setService(int Service) {
    this.Service = Service;
}

public int getFuncionario() {
    return Funcionario;
}

public void setFuncionario(int Funcionario) {
    this.Funcionario = Funcionario;
}

public String getIdentificacao() {
    return Identificacao;
}

public void setIdentificacao(String Identificacao) {
    this.Identificacao = Identificacao;
}

public String getTelemovel() {
    return Telemovel;
}

public void setTelemovel(String Telemovel) {
    this.Telemovel = Telemovel;
}

public String getStart() {
    return Start;
}

public void setStart(String Start) {
    this.Start = Start;
}

public String getEnd() {
    return End;
}

public void setEnd(String End) {
    this.End = End;
}
```

```
public boolean isNotificado() {
    return Notificado;
}

public void setNotificado(boolean Notificado) {
    this.Notificado = Notificado;
}

public String getDate() {
    return Date;
}

public void setDate(String Date) {
    this.Date = Date;
}

public int getEspera() {
    return Espera;
}

public void setEspera(int Espera) {
    this.Espera = Espera;
}

public String getSendStatus() {
    return SendStatus;
}

public void setSendStatus(String SendStatus) {
    this.SendStatus = SendStatus;
}

public String getSmsId() {
    return SmsId;
}

public void setSmsId(String SmsId) {
    this.SmsId = SmsId;
}

public String getSendCode() {
    return SendCode;
}

public void setSendCode(String SendCode) {
    this.SendCode = SendCode;
}

public String getSendMessage() {
    return SendMessage;
}

public void setSendMessage(String SendMessage) {
    this.SendMessage = SendMessage;
}

public boolean isAlerta() {
    return Alerta;
```

```
    }

    public void setAlerta(boolean Alerta) {
        this.Alerta = Alerta;
    }

    public boolean isPrioridade() {
        return Prioridade;
    }

    public void setPrioridade(boolean Prioridade) {
        this.Prioridade = Prioridade;
    }

    @Override
    public String toString() {
        return "Senha{" + "ID=" + ID + ", NumAtendimanto=" + NumAtendimanto + ", Service=" + Service +
", Funcionario=" + Funcionario + ", Identificacao=" + Identificacao + ", Prioridade=" + Prioridade + ",
Telemovel=" + Telemovel + ", Start=" + Start + ", End=" + End + ", Notificado=" + Notificado + ", Date=" +
Date + ", Espera=" + Espera + ", SendStatus=" + SendStatus + ", SmsId=" + SmsId + ", SendCode=" +
SendCode + ", SendMessage=" + SendMessage + ", Alerta=" + Alerta + '}';
    }

}
```

[FIM DA CLASSE SENHA]

[INICIO DA CLASSE USER]

```
public class User {

    private int ID;
    private boolean Administrador;
    private String UserName;
    private String Email;
    private boolean Active;

    public int getID() {
        return ID;
    }

    public void setID(int ID) {
        this.ID = ID;
    }

    public boolean isAdministrador() {
        return Administrador;
    }

    public void setAdministrador(boolean Administrador) {
        this.Administrador = Administrador;
    }

    public String getUserName() {
        return UserName;
    }

    public void setUserName(String UserName) {
```

```
        this.UserName = UserName;
    }


    public String getEmail() {
        return Email;
    }

    public void setEmail(String Email) {
        this.Email = Email;
    }

    public boolean isActive() {
        return Active;
    }

    public void setActive(boolean Active) {
        this.Active = Active;
    }

    @Override
    public String toString() {
        return "User{" + "ID=" + ID + ", Administrador=" + Administrador + ", UserName=" + UserName + ",
Email=" + Email + ", Active=" + Active + '}';
    }

}
```

[FIM DA CLASSE USER]

[INICIO DA CLASSE UTENTE]

```
public class Utente {
    private int Id;
    private String Telemovel;
    private String Codigo;

    public int getId() {
        return Id;
    }

    public void setId(int Id) {
        this.Id = Id;
    }

    public String getTelemovel() {
        return Telemovel;
    }

    public void setTelemovel(String Telemovel) {
        this.Telemovel = Telemovel;
    }

    public String getCodigo() {
        return Codigo;
    }

    public void setCodigo(String Codigo) {
```

```
        this.Codigo = Codigo;
    }

}
```

[FIM DA CLASSE UTENTE]

[INICIO DA CLASSE SERVICO]

```
public class Servico {

    private int ID;
    private String Nome;
    private String Identificacao;
    private boolean Active;

    public int getID() {
        return ID;
    }

    public void setID(int ID) {
        this.ID = ID;
    }

    public String getNome() {
        return Nome;
    }

    public void setNome(String Nome) {
        this.Nome = Nome;
    }

    public String getIdentificacao() {
        return Identificacao;
    }

    public void setIdentificacao(String Identificacao) {
        this.Identificacao = Identificacao;
    }

    public boolean isActive() {
        return Active;
    }

    public void setActive(boolean Active) {
        this.Active = Active;
    }

    @Override
    public String toString() {
        return "Servico{" + "ID=" + ID + ", Nome=" + Nome + ", Identificacao=" + Identificacao + ", Active="
+ Active + '}';
    }

}
```

[FIM DA CLASSE SERVICO]

[INICIO DA CLASSE BALCAO]

```
public class Balcao {

    private int ID;
    private String Nome;
    private String Identificacao;

    public int getID() {
        return ID;
    }

    public void setID(int ID) {
        this.ID = ID;
    }

    public String getNome() {
        return Nome;
    }

    public void setNome(String Nome) {
        this.Nome = Nome;
    }

    public String getIdentificacao() {
        return Identificacao;
    }

    public void setIdentificacao(String Identificacao) {
        this.Identificacao = Identificacao;
    }
}
```

[FIM DA CLASSE BALCAO]

[INICIO DA CLASSE SERVICO_BALCAO]

```
public class Servico_Balcao {

    private int ID;
    private int Balcao;
    private int Service1;
    private int Service2;
    private int Service3;
    private int Funcionario;


    public int getID() {
        return ID;
    }

    public void setID(int ID) {
        this.ID = ID;
    }

    public int getBalcao() {
        return Balcao;
    }
```

```
    public void setBalcao(int Balcao) {
        this.Balcao = Balcao;
    }

    public int getService1() {
        return Service1;
    }

    public void setService1(int Service1) {
        this.Service1 = Service1;
    }

    public int getService2() {
        return Service2;
    }

    public void setService2(int Service2) {
        this.Service2 = Service2;
    }

    public int getService3() {
        return Service3;
    }

    public void setService3(int Service3) {
        this.Service3 = Service3;
    }

    public int getFuncionario() {
        return Funcionario;
    }

    public void setFuncionario(int Funcionario) {
        this.Funcionario = Funcionario;
    }
}
```

[FIM DA CLASSE SERVICO_BALCAO]

[INICIO DA CLASSE SENDSMS]

```
public class SendSMS {

    public ResponseSendSMS send(Gateway_config config,int template, String contacto, String message)
throws MalformedURLException, IOException {

        if (config.getUserName() != null && config.getUserKey() != null && template != 0) {
            //URL DE ENCAMINHAMENTO DA MENSAGEM
            String urlString = config.getUrl();
            //SUBSTITUIR OS ESPAÇOS DA MSG PARA +
            message = message.replace(" ", "+");
            //CODIFICAR O SINAL "+" EM CASO DE FORMATO INTERNACIONAL
            contacto = contacto.replace("+", "%2B");
            //os parametros a serem enviados
            Properties parameters = new Properties();
            parameters.setProperty("User", config.getUserName());
            parameters.setProperty("UserKey", config.getUserKey());
```

```
        parameters.setProperty("Template", String.valueOf(template));
        parameters.setProperty("Destination", contacto);
        parameters.setProperty("Msg", message);
        // o iterador, para criar a URL
        Iterator<Object> i = parameters.keySet().iterator();
        // o contador
        int counter = 0;
        // enquanto ainda existir parametros
        while (i.hasNext()) {
           // pega o nome da variavel
           String name = (String) i.next();
           // pega o valor da variavel
           String value = parameters.getProperty(name);

           // adiciona com um conector (? ou &)
           // o primeiro é ?, restantes &
           urlString += (++counter == 1 ? "?" : "&")
                 + name
                 + "="
                 + value;
        }
        System.out.println(urlString);
        // cria o objeto url
        URL url = new URL(urlString);
        // cria o objeto httpurlconnection
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        // conecta com a url destino
        connection.connect();
        // abre a conexão pra input
        BufferedReader br = new BufferedReader(new InputStreamReader(connection.getInputStream()));
        // le ate o final
        StringBuilder newData = new StringBuilder();
        String s = "";
        while (null != (s = br.readLine())) {
           newData.append(s);
        }
        br.close();
        if (new String(newData) != null) {
           //System.out.println(new String(newData));
           ResponseSendSMS resposta = new ResponseSendSMS();
           JSONObject obj = (JSONObject) JSONValue.parse(new String(newData));
           resposta.setStatus(obj.get("status").toString());
           resposta.setCode(obj.get("code").toString());
           resposta.setMessage(obj.get("message").toString());
           resposta.setSMSid(obj.get("msgId").toString());
           System.out.println(resposta.toString());
           return resposta;
        }
     } else {
        System.err.println("SMSconfig invalido!");
     }
     return null;
  }
```

[FIM DA CLASSE SENDSMS]

[INICIO DA CLASSE CONFIGURACOES]

```
public class Configuracoes {
    private int Id;
    private int Prior_espera;

    public int getId() {
        return Id;
    }

    public void setId(int Id) {
        this.Id = Id;
    }

    public int getPrior_espera() {
        return Prior_espera;
    }

    public void setPrior_espera(int Prior_espera) {
        this.Prior_espera = Prior_espera;
    }

}
```

[FIM DA CLASSE CONFIGURACOES]

[INICIO DA CLASSE EMPRESA]

```
public class Empresa {
    private int Id;
    private String Nome;
    private String Abreviatura;

    public int getId() {
        return Id;
    }

    public void setId(int Id) {
        this.Id = Id;
    }

    public String getNome() {
        return Nome;
    }

    public void setNome(String Nome) {
        this.Nome = Nome;
    }
```

[FIM DA CLASSE EMPRESA]

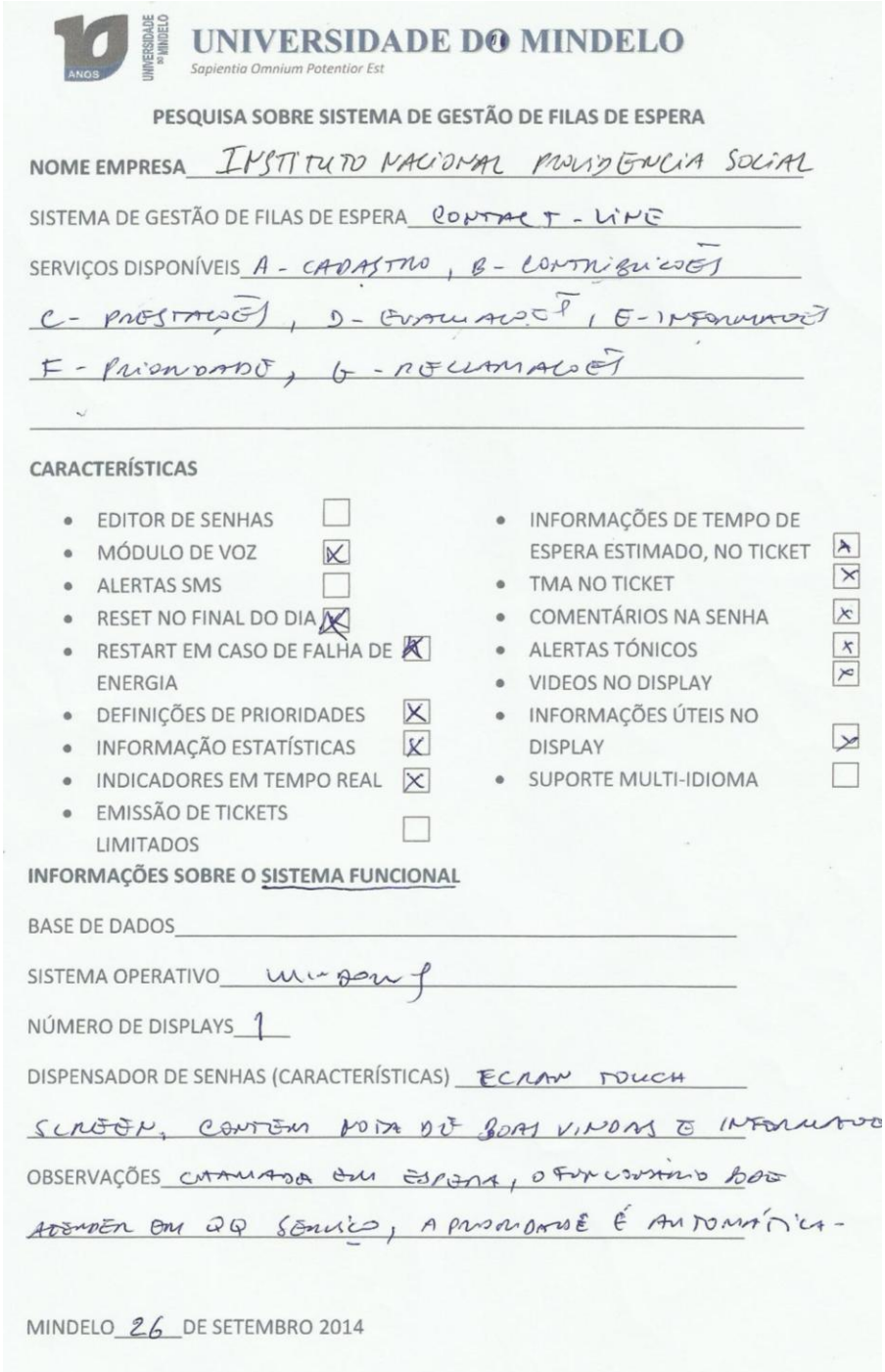
[INICIO DA CLASSE GATEWAY_CONFIG]

```
public class Gateway_config {

    private String UserName;
```

```
    private String UserKey;
    private String Url;

    public String getUserName() {
        return UserName;
    }

    public void setUserName(String UserName) {
        this.UserName = UserName;
    }

    public String getUserKey() {
        return UserKey;
    }

    public void setUserKey(String UserKey) {
        this.UserKey = UserKey;
    }

    public String getUrl() {
        return Url;
    }

    public void setUrl(String Url) {
        this.Url = Url;
    }

}
```

[FIM DA CLASSE GATEWAY_CONFIG]

Modelo utilizado para obter as informações de alguns sistemas de filas



UNIVERSIDADE DO MINDELO
Sapientia Omnium Potentior Est

PESQUISA SOBRE SISTEMA DE GESTÃO DE FILAS DE ESPERA

NOME EMPRESA INSTITUTO NACIONAL PREVIDÊNCIA SOCIAL

SISTEMA DE GESTÃO DE FILAS DE ESPERA CONTACT - LINE

SERVIÇOS DISPONÍVEIS A - CADASTRO, B - CONTRIBUIÇÕES
C - PRESTAÇÕES, D - AVALIAÇÕES, E - INFORMAÇÕES
F - PRIORIDADE, G - RECLAMAÇÕES

CARACTERÍSTICAS

- EDITOR DE SENHAS ☐
- MÓDULO DE VOZ ☒
- ALERTAS SMS ☐
- RESET NO FINAL DO DIA ☒
- RESTART EM CASO DE FALHA DE ENERGIA ☒
- DEFINIÇÕES DE PRIORIDADES ☒
- INFORMAÇÃO ESTATÍSTICAS ☒
- INDICADORES EM TEMPO REAL ☒
- EMISSÃO DE TICKETS LIMITADOS ☐
- INFORMAÇÕES DE TEMPO DE ESPERA ESTIMADO, NO TICKET ☒
- TMA NO TICKET ☒
- COMENTÁRIOS NA SENHA ☒
- ALERTAS TÓNICOS ☒
- VIDEOS NO DISPLAY ☒
- INFORMAÇÕES ÚTEIS NO DISPLAY ☒
- SUPORTE MULTI-IDIOMA ☐

INFORMAÇÕES SOBRE O SISTEMA FUNCIONAL

BASE DE DADOS

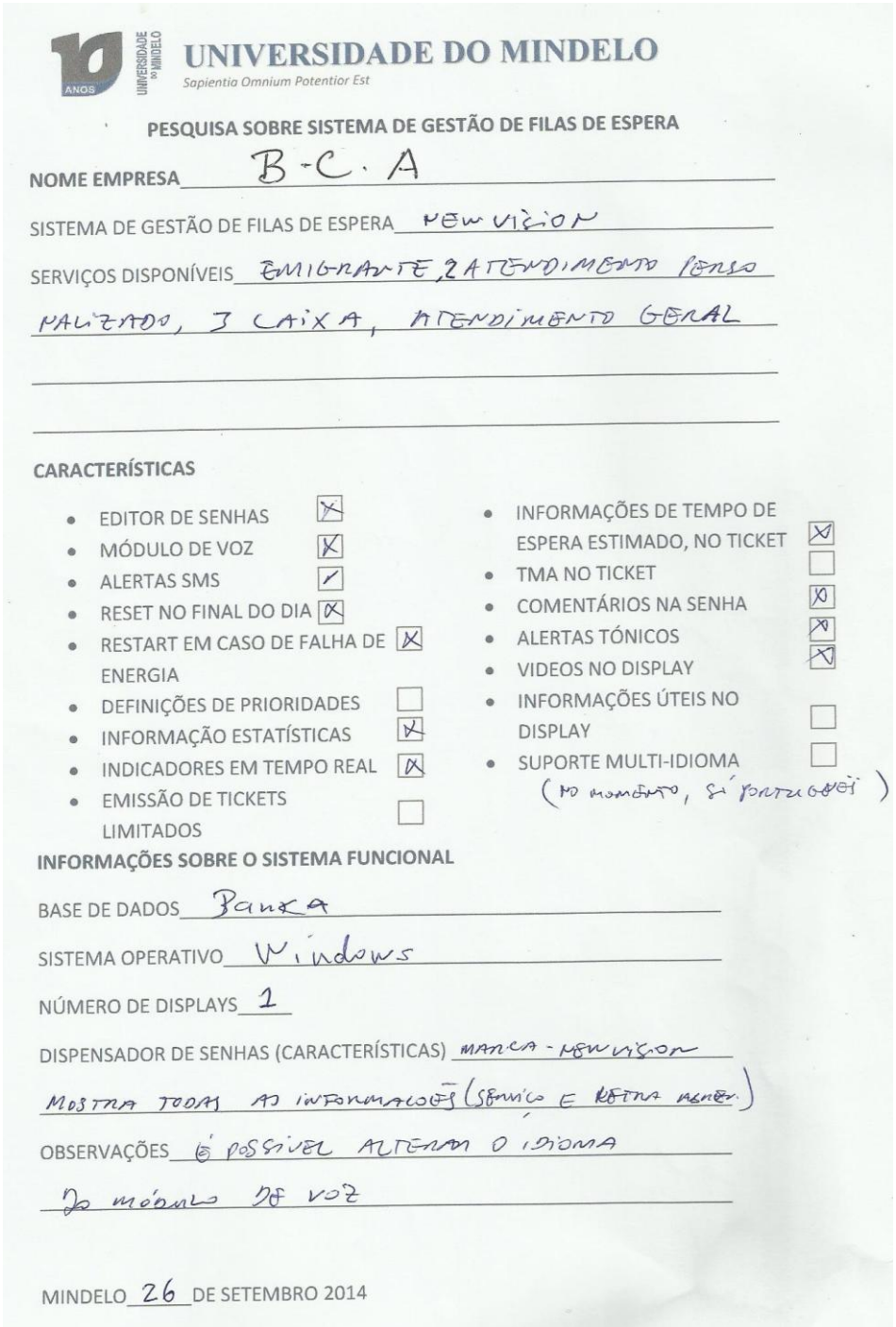
SISTEMA OPERATIVO Windows

NÚMERO DE DISPLAYS 1

DISPENSADOR DE SENHAS (CARACTERÍSTICAS) ECRAN TOUCH SCREEN, CONTÉM MSG DE BOAS VINDAS E INFORMAÇÃO

OBSERVAÇÕES CHAMADA EM ESPERA, O FUNCIONÁRIO DOS ADENDEN OU SERVIÇO, A PRIORIDADE É AUTOMÁTICA -

MINDELO 26 DE SETEMBRO 2014



UNIVERSIDADE DO MINDELO
Sapientia Omnium Potentior Est

PESQUISA SOBRE SISTEMA DE GESTÃO DE FILAS DE ESPERA

NOME EMPRESA B-C. A

SISTEMA DE GESTÃO DE FILAS DE ESPERA NEW VISION

SERVIÇOS DISPONÍVEIS EMIGRANTE, 2 ATENDIMENTO PÓNSO PALIZADO, 3 CAIXA, ATENDIMENTO GERAL

CARACTERÍSTICAS

- EDITOR DE SENHAS [X]
- MÓDULO DE VOZ [X]
- ALERTAS SMS [✓]
- RESET NO FINAL DO DIA [X]
- RESTART EM CASO DE FALHA DE ENERGIA [X]
- DEFINIÇÕES DE PRIORIDADES [ ]
- INFORMAÇÃO ESTATÍSTICAS [X]
- INDICADORES EM TEMPO REAL [X]
- EMISSÃO DE TICKETS LIMITADOS [ ]
- INFORMAÇÕES DE TEMPO DE ESPERA ESTIMADO, NO TICKET [X]
- TMA NO TICKET [ ]
- COMENTÁRIOS NA SENHA [X]
- ALERTAS TÓNICOS [X]
- VIDEOS NO DISPLAY [X]
- INFORMAÇÕES ÚTEIS NO DISPLAY [ ]
- SUPORTE MULTI-IDIOMA [ ] (NO MOMENTO, SÓ PORTUGUÊS)

INFORMAÇÕES SOBRE O SISTEMA FUNCIONAL

BASE DE DADOS Banca

SISTEMA OPERATIVO Windows

NÚMERO DE DISPLAYS 1

DISPENSADOR DE SENHAS (CARACTERÍSTICAS) MARCA - NEW VISION

MOSTRA TODAS AS INFORMAÇÕES (SERVIÇO E HORA MESM.)

OBSERVAÇÕES É POSSIVEL ALTERAR O IDIOMA DO MÓDULO DE VOZ

MINDELO 26 DE SETEMBRO 2014